UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Master in Deep Learning for Audio and Video Signal Processing

MASTER THESIS

ON EXPLORING THE USE OF SYNTHETIC DATA FOR SEMANTIC SEGMENTATION IN VIDEOS

Roberto Alcover Couso
Tutor: Juan Carlos San Miguel Avedillo

July 2021

ON EXPLORING THE USE OF SYNTHETIC DATA FOR SEMANTIC SEGMENTATION IN VIDEOS

Roberto Alcover Couso
Tutor: Juan Carlos San Miguel Avedillo

Video Processing and Understanding Lab
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
July 2021

Trabajo parcialmente financiado por el Gobierno de Espana bajo el proyecto TEC 2017-88169-R (MobiNetVideo).

Abstract

Since the rise in popularity of deep learning with the ImageNet challenge, where it was proven that with sufficient data neural networks could outperform traditional algorithms in simple tasks. The community agreed that the main problem to solve is how to achieve reasonable performance when there is not sufficient data. Whether it comes from lack of viability (e.g. autonomous driving when human lives are at stake) or resources, due to the labelling costs, efficiently capturing enough data remains unsolved. In this context, we study the possibility of using synthetic data from simulators in order to train deep neural networks for semantic segmentation. Using state of the art domain shift algorithms, we train models with few or no real data, exploiting the unlimited labelled data provided from simulators. Our results suggest that the inclusion of synthetic data from simulators improves performance in different tasks where there is little real data. Furthermore, we find that prior training with only synthetic data as a weight initialization, leads to a significant performance increase compared to training with only real data. However, it comes with an increased need of training time as a larger dataset is employed, and the performance is highly correlated to the quality of the simulator, therefore specific synthetic generators need to be made for complex tasks. From ImageNet we inferred that when sufficient data is provided we can produce disrupting changes, and this work suggests that with simulators that are good enough any problem can be tackled

Keywords

Synthia, Mapilliary, Kitti, Cityscapes, Deeplab, MSS, deep neural networks, domain shift, transfer learning, domain gap, dataset, synthetic data, simulator, semantic segmentation, real data, training

Acknowledgment

I want to thank my family for the constant support. My sister Veronica, for always staying by my side even in my worst days, pushing me to be the best version of myself. My parents for the unconditional love and helping me through this stressing period.

My friends Celia, Pipo and Alvaro for the laughs, drinks and relaxing moments. Those mental resets were truly needed.

My tutor, Juan Carlos, who was every week helping me throughout the process.

Finally i'd like to thank to all those artist who entertained me through out this process. Tried to be nonchalant about most of the failures, underneath a guise of a smile i just wanted you to know that this is me trying.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we introduce and motivate the context of the Master Thesis project. Starting by giving a brief motivation of the project and it's importance, we present the main goals of the project describing some of the challenges and the expected results, concluding with the project organization, describing briefly each of the chapters of this report.

1.1 Motivation

Since the beginning of last decade, the field of Computer Vision has experienced an abrupt evolution, where visual classifiers based on handcrafted features were the main focus. During this period, the research strategy addressed two main challenges. First, finding discriminative visual descriptors, such as Haar wavelets [1], SIFT[2], LBP [3], or HOG [4]. Second, once the descriptor was chosen, machine learning algorithms were implemented to use those descriptors as features such as SVM [5] and random forest [6], aiming at finding class borders to discriminate in the feature space between classes.

However, with the rose of popularity of deep learning with the ImageNet triumph [7], when a neural network was successfully implemented to automatically discriminate which features are more relevant for the desired task through nested convolutions and gradient descent. Having the paradigm shifted, further efforts were focused on finding better architectures to abstract richer features. As these architectures evolved, the community agreed that with enough data any problem could be tackled, therefore the focus was set into achieving reasonable performance when there is not sufficient data.

Despite these advances many problems remain unsolved mainly due to insufficient data: either the available datasets are too small or, also very often, even while capturing unlabeled data is relatively easy, the costs of manual labeling are prohibitively high.

For example, let us consider semantic segmentation, a typical Computer Vision problem. Semantic segmentation is the task of labeling each pixel of an image. In order to produce a labeled dataset, at some point, all images must be manually processed, whether it is by a pixel by pixel level or a refinement of geometrical predictions. Furthermore, human verification of each image must be made to ensure correct segmentation masks.

As a response to this limitation for annotating data, many efforts have shifted the focus onto synthetic data as a plausible solution, having developers creating a 3D environments with models of the objects to recognize. The background and surroundings can be rendered [8] or an overlay of 3D models with real images [9]. While 3D mod-

eling is still mostly a manual labor, it can potentially provide unlimited amounts of labeled data, not only RGB images and segmentation maps but also depth images and synthetic video clips to name a few.

Based on the *MSS* simulator [10][11][12], developed by the VPU lab from the Autonomus University of Madrid. This simulator provides unlimited synthetic images and their corresponding ground truth in an urban scenes scenario. *MSS* is an open source simulator to generate automatically labeled synthetic urban scenes based on Unity. The simulator is developed in a virtual city where 12 distinct elements are present. This resource separates itself from similar ones by giving the user the freedom to generate as many images as needed, to select the position, angle and brightness of the camera to capture the frames. Due to the flexibility of the simulator and the amount of data that one can generate, we believe this simulator can bring great advances to the field.

Although promising, using synthetic data does not come without inherent problems, mainly the domain gap between synthetic and real data. Synthetic images present different visual appearance than real images, therefore efforts must be made in order to alleviate this problem, two main approaches can be considered.

One is having efforts focused on generating images as photorealistic as possible [[13],[14],[15],[16]]. However, this is mostly correlated with current rendering advances or style transfer [16][13] with the possible hallucinations of the network, seen in works like [14].

Second is implementing techniques to transfer learning obtained from the synthetic data to the real domain [[17][18][9][19][20][21][22]]. Consequently, domain adaptation is a major topic in synthetic data research and the chosen approach of this project.

Based on the good results obtained in similar fields, such as object detection [[17][9][18][19]], and the intrinsic difficulties of labelling a semantic segmentation dataset, makes synthetic datasets a useful resource for this challenge.

The project aims at studying the impact and problems of using synthetic data from simulators in order to train neural networks in the semantic segmentation task. Specially exploring the usage of the *MSS* simulator.

1.2 Goals

The goal of this project is to identify if synthetic data from the *MSS* simulator can be useful in semantic segmentation and to what extent. With this goal in mind we find different secondary objectives of the project:

- The state of the art. Learning the latest advances in the topic and how the scientific community is currently trying to solve the remaining challenges. With focus in the domain shift problem between real and synthetic images.

- Gather of real and synthetic datasets. In order to understand whether or not the data from the *MSS* simulator is useful, we need a baseline of the current state of the art and it's performance. In order to provide a fair comparison we need to gather popular real and synthetic datasets to evaluate under the same conditions.

- Framework to ease the training process of different networks and different datasets. In line with the previous goal, we need to dynamically mix and join real and synthetic data to train and perform different experiments. This mixture must not

be handmade. In order to avoid introducing new biases from the data selection a random selection must be used.

Additionally, replicability must be ensured in the scientific research. Consequently a random seed should be easy to insert.

- Design criteria for synthetic data in the *MSS* simulator. As this simulator provides unlimited data with complete freedom of point of view and movement in a 3D virtual city. We need to discriminate what makes a good sequence to extrapolate knowledge to real scenarios.

  Once we have established what makes a good sequence, generate a full synthetic dataset using the *MSS* simulator.

- Analyze the impact of using the generated dataset and other synthetic datasets in the performance of state of the art models tested on real images. Different performance metrics are used in order to measure the impact of including synthetic images in the training.

  Our main goal is finding a significant difference in the performance of models which have been trained with synthetic data and models trained only with real data. Furthermore, analyze in a per-class basis the impact of synthetic data and lay a strong foundation for future works to abstract from the results.

1.3 Project structure

This project consists of the following chapters:

- chapter 1 Introduction.

  In this chapter we give a brief introduction to the project. Why should it be of any interest and the expected results of the project.

- chapter 2 State of the art.

  In this chapter we go through how synthetic data is being used in the semantic segmentation and the rising approaches to make use of synthetic data. Making special emphasis on different approaches to tackle their inherent difficulties.

- chapter 3 Design of data sequences.

  In this chapter we analyze the impact of different sequences in the training of synthetic algorithms. Starting from a set of sample sequences we analyze and dissect how the model learns when including these sequences. From this knowledge we conclude with the generation and analysis of a full dataset using the *MSS* simulator.

- chapter 4 Evaluation Methodology.

  In this chapter we explain the project development. Going from the testing environment, including the software, the methodology, the selected datasets and the performance evaluation metrics used to compare results.

- chapter 5 Experimental Results.

  In this chapter we discuss the performed experiments and the results obtained. This chapter serves as a closing act for the work performed compressed into a set of experiments and results.

- chapter 6 Conclusions and future work.

  This chapter includes the end summary of the work. Finally the Master thesis is wrapped with the future work.

Chapter 2

State of the art

In this chapter we will introduce the related work in Computer Vision where synthetic data is used. Due to the close relation between object detection and semantic segmentation we included a brief survey on object detection in Appendix A. Guided by the good results in object detection, the work continues to a new problem, semantic segmentation. We analyze how it is being tackled, the specific difficulties which rises in semantic segmentation, the main architectures used in semantic segmentation and current approaches with synthetic data to solve this problem. Finally, we conclude comparing different synthetic datasets for semantic segmentation.

2.1 Main ideas from object detection

As this topic is not the core of this project, we included in the Appendix A the current state of the art of object detection. In this section we present only a brief summary of the conclusions gathered from the papers. For further explanation go to Appendix A

Domain shift is defined as: the scenario where the training and test data are not sampled independently from an identical distribution, i.e. i.i.d assumption is not fulfilled. The domain shift always brings a drop in performance if not tackled correctly [23], [17],[18].

Domain shift evidently occurs between synthetic and real data, but also appears when different sources of real data are employed. From [17] we can extrapolate that although the best results are obtained when the i.i.d. assumption is fulfilled, there is a drastic drop in performance whether the source domain is real or synthetic. This means that the domain gap persist even between real datasets, thus, domain adaptation is needed nevertheless.

Using as starting weights a previously trained network with synthetic data outperforms training with just real data. In [17] authors illustrate how consistently training from pretrained weights on synthetic data and finetuning with a the real dataset yields better results than training with only real data. Furthermore, they illustrate how training with as little as 10% of the real data, in conjunction with synthetic data, can potentially generate models which compete with the ones trained with the whole real dataset.

Having access to unlimited amounts of synthetic data allows to train smarter. Providing models which can compete with the ones trained with real data. In [19] authors

prove that the results obtained using curriculum learning on purely synthetic data yields better models than a synthetic data overlayed on real data, and comparable with the ones obtained by training with up to 5000 real images.

2.2 Semantic segmentation

From [17] and [19] authors showcase the advantages of using synthetic data in object detection to outperform training with only real data. Therefore, we will analyze the state of the art in a different task, semantic segmentation. Here the usage of synthetic data is still an open question.

Semantic segmentation is the task of labelling each pixel of an image with a corresponding class of the semantic object represented, see Figure 2.1. Simply, our goal is to take either a RGB color image $\in \mathrm{R}^{height \times width \times 3}$ or a grayscale image $\in \mathrm{R}^{height \times width \times 1}$ and output a segmentation map where each pixel contains a class label represented as an integer $height \times width \times 1$.

From the definition, we can follow by affirming that the network is in a way an injective function, which maps from the RGB domain to the label domain. It is injective because two identical scenes with different lighting conditions have different RGB representation, yet their corresponding mapping must be equal.

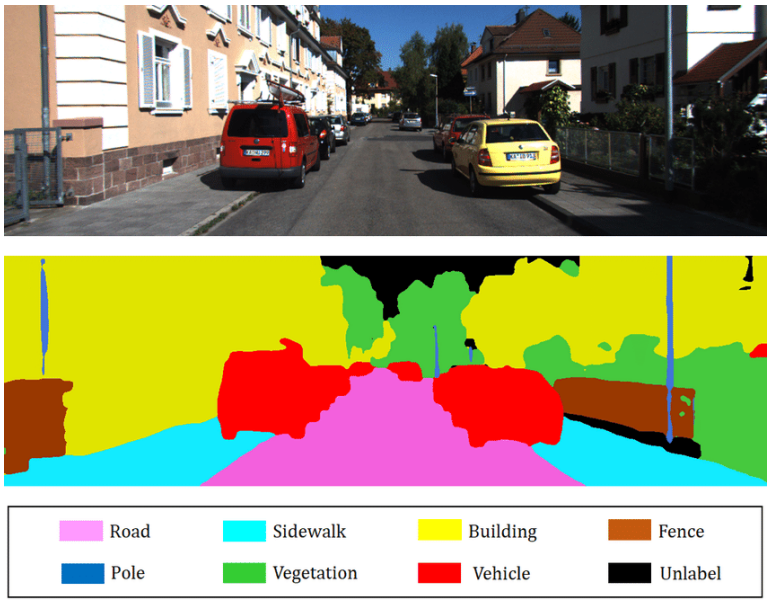


Figure 2.1: Example of semantic segmentation. [24]

2.2.1 Main algorithms in deep learning

With a first glance at the problem one may intend to use convolutions from start to end without downsampling. Mainly due to the nature of the problem being an injective mapping between two matrix with the same dimenssionality. However, to the best of our knowledge, the only successful attempt is [25]. Mainly due to the computational cost of performing convolutions at high resolution being prohibitively high to be feasible by current hardware. Therefore the main approach is to downsample the image to a smaller feature space with convolutions, followed by an upsampling stages.

The first algorithm we will analyze is *Fully convolutional network*, [26], similar to what *Single Shot MultiBox detector*, *SSD*, proposed (See object detection Appendix A), this algorithm proposes to combine layers of the feature hierarchy refining the spatial precision of the output. *FCNs* followed the idea that semantic information refines as we go deeper. However, spatial information gets fuzzy and lost the deeper we go. Therefore, combining shallower layers, providing stronger spatial information and deeper layers providing richer semantic information. see Figure 2.2.

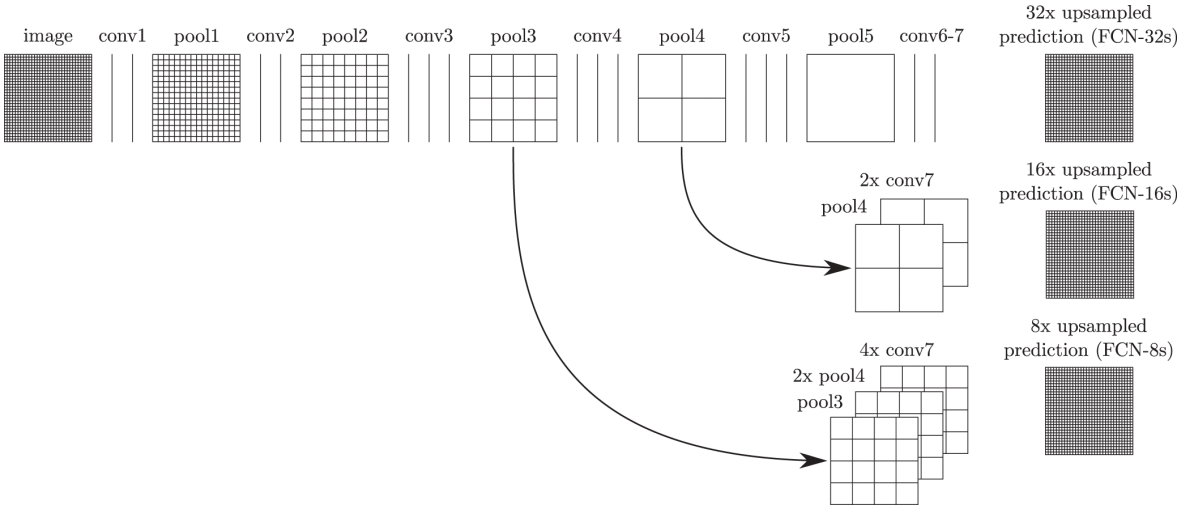


Figure 2.2: FCN combination of layers. [26]

.

A different approach describes *DeepLabv3*, [27] where the authors included the idea of *atrous/dilated* convolution, aiming to expand the receptive field of a filter. This convolution is a dilation of the normal convolution, in other words a convolution with holes, thus, the name of atrous, which in French means holes. The introduction of these holes yield an increasing field of view by the number of holes in the convolution, see Figure 2.3.

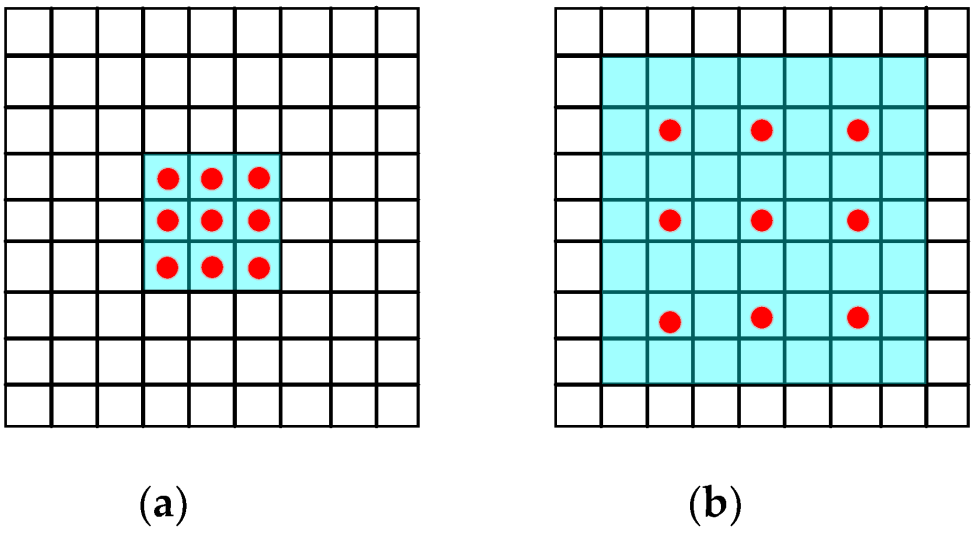


Figure 2.3: [28]. The subfigures represent the samples taken for performing convolutions for (a) normal convolution and (b) dilated convolution with a factor of 2

Following the idea of *Spatial Pyramid Pooling*, mentioned previously with *SPN* (See object detection Appendix A.1).*DeepLabv3* included *Atrous Spatial Pyramid Pooling* (ASPP) to robustly segment objects at multiple scales with filters at multiple sampling rates and effective fields-of-views, using *atrous convolution* with different dilation ratios, see Figure 2.4.

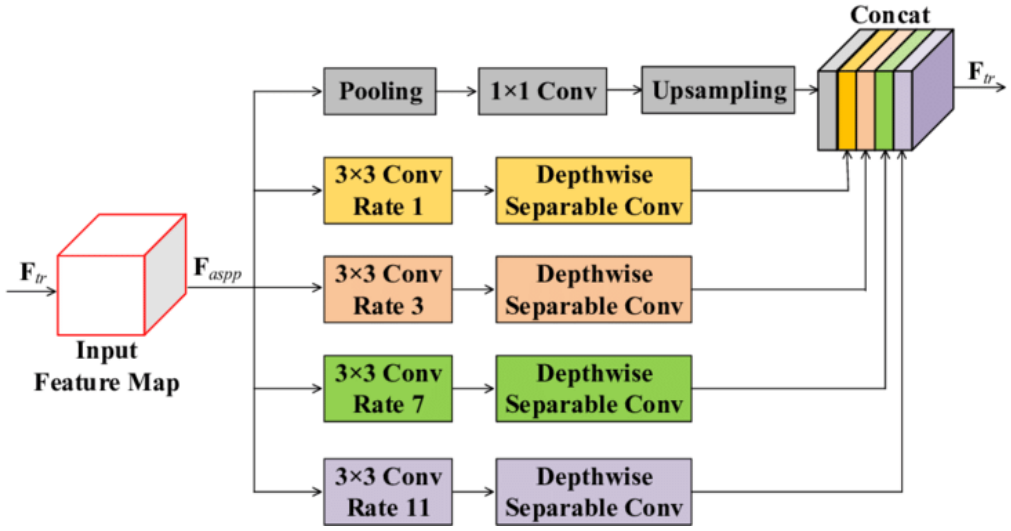


Figure 2.4: Atrous spatial pyramid pooling [27]

.

2.2.2 Synthetic data in Semantic Segmentation

The introduction and usage of synthetic data in the semantic segmentation field is still an evolving topic. In addition, due to the efforts needed to label a huge dataset, the complexity of the task, and the lack of an staple semantic segmentation dataset there is no unified opinion on how to proceed. We find many approaches being followed to solve this task. We propose two main branches.

First, handling the domain shift as a problem to be solved through the learning protocol of the network by *domain adaptation*. A basic assumption in machine learning is having the training and test data sampled independently from an identical distribution, i.e. i.i.d assumption. In contrast to using synthetic data, where the training/source data comes from one source and the test/target data comes from another source. This domain shift leads to a significant performance drop on the test set. Domain adaptation aims to alleviate the impact of such distribution mismatch. Reinforcing the generalization ability of the learned model through techniques such as: previous learning general tasks [29] and using already trained models to guide the learning, hence, forcing the network to be unaffected by the domain shift through the loss [30].

Second, handling the domain shift as a *data problem*, the goal with this approach is to generate a synthetic dataset as photorealistic as possible, thus, removing the problem from the source. This approach usually relies on style transfer [16] and GANS [13] to generate a new refined dataset. This new dataset is then used, usually in combination with real data, to train a semantic segmentation network.

Handling domain shift through domain adaptation

When handling the domain shift one concept is key, knowledge distillation [31]. Knowledge distillation is the technique of transferring knowledge from a trained model (usually bigger), teacher, to another model (usually smaller), student. The distillation loss is usually used in conjunction with the standard loss to match the GT. The goal is to match the logits from the teacher model, hence mimicking it's behaviour. The logits determine how a teacher's knowledge is captured. Therefore, reinforcing through the loss to preserve similar activation patterns to the teacher, will yield to transferring knowledge to the student, see Figure 2.5.

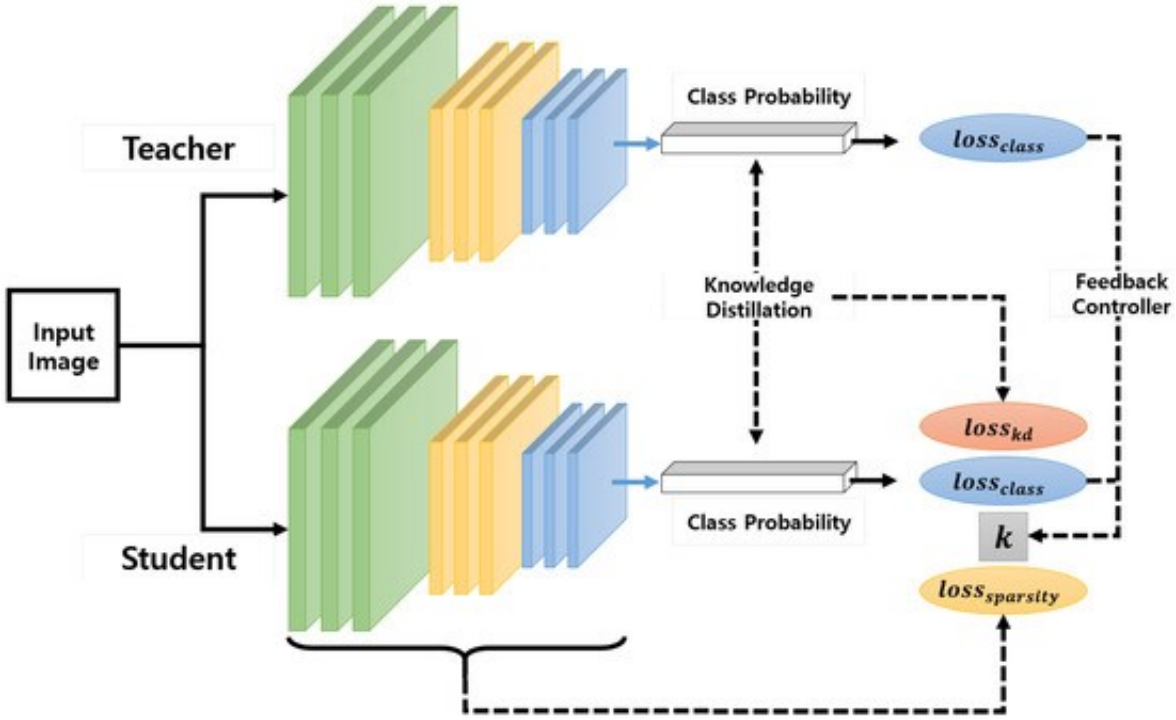


Figure 2.5: Teacher student architecture from [32]

Three main approaches to handle the domain shift as a *data problem* can be classified:

First, *Curriculum learning* [21]: In this paper the authors propose a pretext task, super pixel, where the network first try to learn the semantic classification of the mode of a cluster of pixels. The GT is divided into different 'super pixels', and each one is classified as the mode of the clusters. Therefore, the curriculum process commences by learning to estimate the global distribution of pixels, then once the network has adapted it's weights to classify the super pixels, which are a generality which should be preserved regardless of the domain, (Synthetic and real images have a similar global distribution) they follow by learning the semantic classification and reinforcing the loss with the super pixel loss, see Figure 2.6.

Second, including distillation loss from a pretrained network on real data [22]: In this work the authors propose to use a pretrained network on real data in order to guide the learning of the new network trained only on synthetic data. This method focus on learning a rich and general representation to tackle both domains jointly by

using real images to guide the learning of the network to learn robust convolutional filters. see Figure 2.7.

Finally, including adversarial loss to adapt synthetic data to real world scenes: The authors of this paper propose to introduce a new network, discriminator, to the problem. The role of the discriminator network is to distinguish the synthetic maps generated from real images and the ones from synthetic images. see Figure 2.8.

Future works like [20], follow by using this approach only with synthetic GT. In this approach the discriminator's loss does not depend on the ground truth of the real images and the classifier can be trained only through synthetic ground truth. This allows to exploit real images in an unsupervised manner to tackle the domain shift problem.

Handling domain shift as a data problem

This approach generally uses a second network to refine or generate new images from synthetic data, these new images have a better resemblance of the real datasets than the ones obtained from simulators. The usual scheme followed by these networks goes as follow:

First, *input adaptation*. From the synthetic data obtained from a simulator and real data train a generator network to output an RGB image. This generated image is compared with real images through a discriminator network.

These networks usually follow the GAN scheme of generator discriminator established by [13], where the training of the generator network is guided by the loss of the discriminator network. The generator, G tries to maximize the $logD(G(z))$ while the discriminator D tries to maximize the $log(D(x)) + log(1 - D(G(z))$, where:

- $D(x)$ is the discriminator estimate of the probability that the real image x is real
- $G(z)$ is the generators output when given synthetic data z.
- $D(G(z))$ is the discriminator estimate of the probability that the generated image is real

Second, *output adaptation*. From the generated images and real images, train a network to perform the semantic segmentation. The GT of the generated images is the same as the original synthetic image.

One existing problem in the output adaptation is that the generator network may hallucinate objects, which as the GT is not modified, will not be present. Consequently, many authors follow the idea from [33] of including a final discriminator network on the predicted segmentation maps to further guide the learning process of the generator, see Figure 2.8.

Handling domain shift as a *data problem* is followed by works like [14], where they propose to transfer the style to real images as well. Authors propose to reinforce the classification network to learn from refinements of real data as well. They refined with a second network real and synthetic images. In a way having three distinct datasets,

refined real images, refined synthetic images and real images. Allowing to divide the loss into each of the subsets and weighting each subset impact on the loss.

The authors of [15] continue this work by including the depth from synthetic images. Alluding to the close relation that semantics and geometry present. This work alleviates the problem of hallucinations from the generator network due to it's additional constraints.

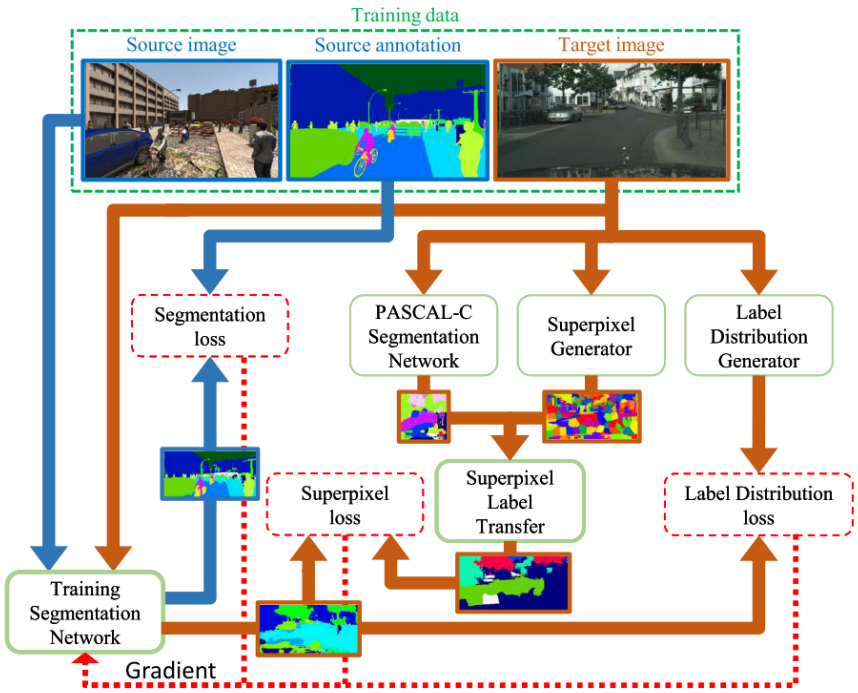


Figure 2.6: Curriculum learning on semantic segmentation, super pixel approach from [21]

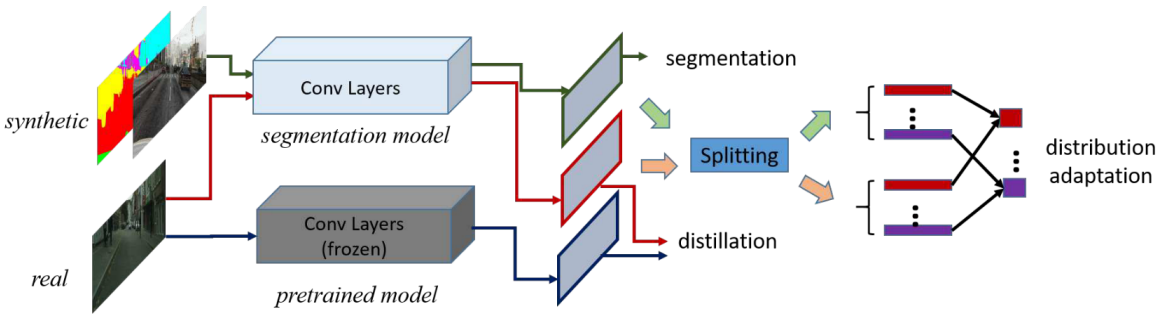


Figure 2.7: Distillation loss on semantic segmentation from [22]

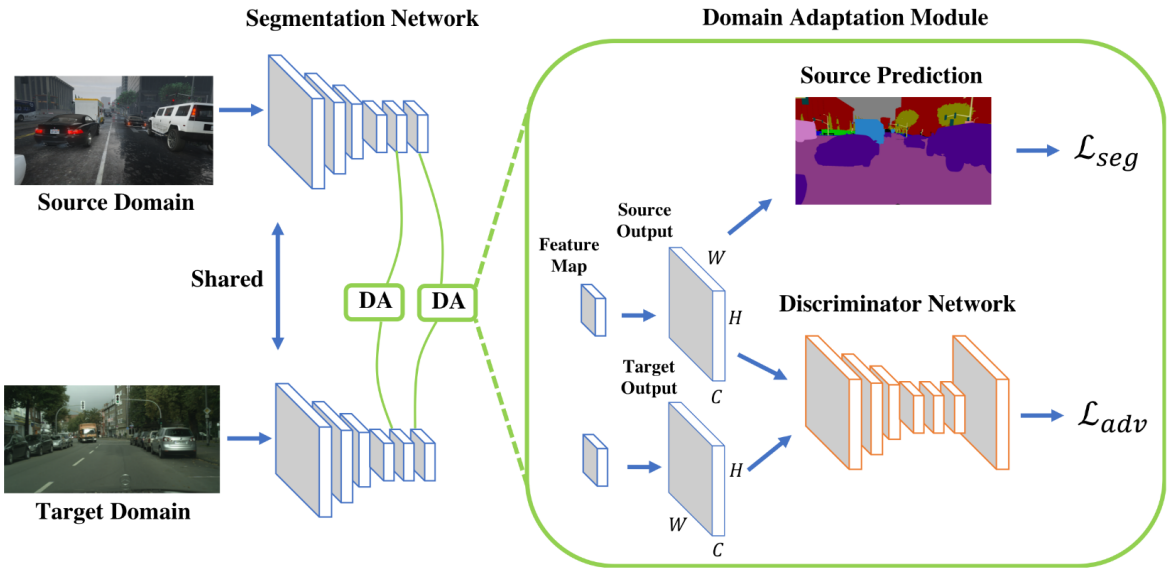


Figure 2.8: Adversarial loss on semantic segmentation with real images from [33]

2.2.3 Conclusions from synthetic data applied to semantic segmentation

In order to summarize the state of the art on the semantic segmentation topic, we make emphasis into three main ideas:

- While synthetic data seems to be useful, in the context of semantic segmentation is still an evolving topic. ;any different approaches are being used in order to tackle it. However, there's not a conclusive argument of whether or not photo realism is a must.

  We can see many works focused on removing/alleviating all discrepancies between real and synthetic images, mainly appearing on the light reflection, color and texture. While at the same time, other trend is being followed, domain randomization, where applying the principles of [9] into the semantic segmentation field. Exploiting the limitless amounts of data synthetic data brings by creating random scenes with little to no restrictions to the objects placement and geometric structure of the scene.

- When following an style transfer approach seems like the community has agreed to follow the architecture presented by [33] and include a discriminator to the output semantic maps to guide the adaptation.

- Combining the losses from different sources seems to affect favorably to the final performance of the model. Whether is the inclusion of the depth into the model like [15], introducing distilation loss from a pretrained model like [22] or adding a pretext task like the superpixel from [21] seem to reinforce the network to focus on the general aspects shared between real and synthetic data alleviating the domain shift.

2.3 Datasets

In this section we analyze the current state of the art real and synthetic datasets for urban scenes semantic segmentation.

2.3.1 Real datasets

This work focuses on urban scenes datasets, we only analyze datasets containing this type of footage. These datasets are composed of different video sequences recorded in street scenes. Usually by placing a camera on a car or a traffic pole and afterwards manually labelling them. In this project we analyze three real datasets.

The *Mapilliary* dataset [34][1]. Mapillary is a community-led service for people who collaboratively want to visualize the world with street-level photos with a minimum size of 1920×1080. Additionally, around 90%of the images were selected from road/sidewalk views in urban areas, the remaining ones are from highways, rural areas and off-road. Given these constraints. This classification was manually evaluated. Furthermore, a minimum quality was required, removing degraded images exhibiting strong motion blur, rolling shutter artifacts, interlacing artifacts, major windshield reflections or containing dominant image parts from the capturing vehicledevice(like car hood, camera mount or wipers).

The *Cityscapes* dataset[35][2]. This dataset is composed by several hundreds of thousands of frames from a moving vehicle acquired during the span of several months, covering spring, summer, and fall in 50 cities, primarily in Germany but also in neighboring countries.

The *Kitti* dataset[36][3]. This dataset was captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. Up to 15 cars and 30 pedestrians are visible per image. Due to its size this dataset will only be used for testing in order to analyze generalization power in conditions not present on training, such really bright scenarios with some overexposure and motion blurriness.

The Semantic Drone Dataset[4]: The Semantic Drone Dataset focuses on semantic understanding of urban scenes for increasing the safety of autonomous drone flight and landing procedures. The imagery depicts more than 20 houses from nadir (bird's eye) view acquired at an altitude of 5 to 30 meters above ground.

2.3.2 Synthetic datasets

Although generating synthetic environments is a challenging task by itself, current real datasets are not able to represent all variability found in a road, thus, the interest in generate synthetic data to complement real datasets. This remark rose as early as 1989, whith ALVINN [37], one of the first autonomous driving attempts based on neural networks. Using 30×32 videos, was already training on synthetic data. The

[1] http://www.mapillary.com
[2] https://www.cityscapes-dataset.com/
[3] http://www.cvlibs.net/datasets/kitti/
[4] https://www.tugraz.at/institute/icg/research/team-fraundorfer/software-media/dronedataset/

Name	Year	Ref	Engine	Notes
TORCS	2014	[39]	Custom	Game-based simulation engine
Virtual KITTI	2016	[36]	Unity	Include synthetic cars in real images
GTAVision	2016	[38]	GTAV	200000 images
SYNTHIA	2016	[8]	Unity	220000 images
VIPER	2017	[40]	GTAV	GTAV based pedestrian recognition
CARLA	2017	[41]	Custom	Simmulator, as a support for development, training, and validation of autonomous driving systems
AADS	2019	[42]	Custom	Augmented Autonomous Driving Simulation.
PreSIL	2019	[43]	GTA V	Synthetic images with point-wise segmentation and depth information.
MSS	2020	[10]	Unity	Open source virtual city which provides the user freedom to capture and generate automatically labelled frames.

Table 2.1: Overview of synthetic datasets and virtual environments of urban scenes

authors remark that "Training on actual road images is logistically difficult, because in order to develop a general representation, the network must be presented with a large number of training exemplars depicting roads under a wide variety of conditions." and described a simulator to tackle that problem. Although over 20 years have passed since ALVINN was published, we find that still real datasets fail to provide the variability and size needed to fully exploit the power of current architectures. This is the reason why synthetic datasets are being used as a staple in semantic segmentation (over 1000 cites of the Synthia dataset in less than 5 years).

We can classify urban and outdoor environments by the engine used to generate them:

- Unity: Unity is a cross-platform game engine developed by Unity Technologies.
- GTA-V: Rendered images from the open-world video game *Grand Theft Auto 5* with an car-egocentric point of view driving through American virtual cities.[38]
- Custom: Many efforst follow domain randomization (see subsection A.1.2), overlaying car models on real backgrounds as a data augmentation approach and using GANS to augment datasets.

Table 2.1 gives a brief overview of current synthetic datasets and virtual environments (Mainly used for reinforcement learning) available for urban scenes simulation.

Most efforts of this work are focused on:

Synthia [8]: The SYNTHetic collection of Imagery and Annotations, is a dataset that has been generated with the purpose of aiding semantic segmentation and related scene understanding problems in the context of driving scenarios. SYNTHIA consists of a collection of photo-realistic frames rendered from a virtual city and comes with precise pixel-level semantic annotations.

MSS: Multi-camera System Simulator (MSS) developed in the TFG of 2017 by Mario González within the VPULab research laboratory of the Universidad Autónoma

de Madrid [12]. MSS is a simulator which provides a complete API for automatically label synthetic scenes from the simulator.

2.3.3 Comparative analysis of datasets

(2.9.1) Cityscapes

(2.9.2) Mapilliary

(2.9.3) Kitti

(2.9.4) The semantic drone dataset

Figure 2.9: Sample image of real datasets. Cityscapes, Mapilliary, Kitti and the Semantic Drone Dataset

In Figure 2.9 we can see how although Cityscapes, Kitti and Mapilliary are 3 real datasets, there is still a visible domain gap between them, due to light conditions, architectural differences and even the biases of each dataset.

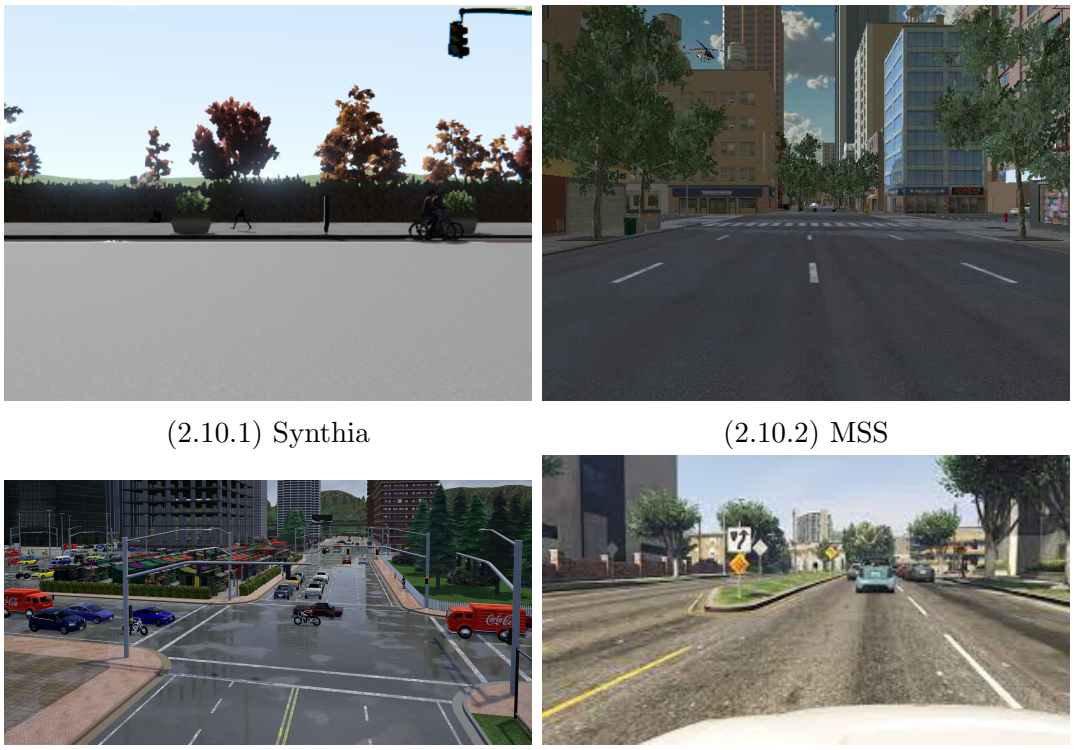
We find that in the Cityscapes dataset, there are little to no poles, due to the city wiring being located underground. In contrast with the Mapilliary dataset, which is obtained from cities where the city wiring tends to be supported by utility poles.

This illustrate that even comparing real datasets there is a consistent domain gap which will need to be tackled in order to use different datasets for training. Hence, supporting the usage of synthetic datasets. If there's a persistent domain gap through datasets tackling the domain gap of synthetic and real images will serve to a tackling the domain gap between real datasets, therefore improving the results achieved with any available data.

In Figure 2.10 we include illustration of synthetic datasets, we can see how even among synthetic environments, each dataset is unique in it's composition.

In Table 2.2 we include some popular datasets for urban scenes. One thing to note is that real datasets do not provide enough images to be used in practice. have too little images to be used in practice.

Some datasets, such as Cityscapes alleviates the costs of labelling semantic segmentation GT by including coarse annotations. These annotations describe global



(2.10.1) Synthia (2.10.2) MSS

(2.10.3) Carla (2.10.4) GTA V

Figure 2.10: Sample image of synthetic datasets. Synthia, MSS, Carla and GTA V.

semantics, while fine grained annotations describe finer details of the image. In general coarse annotations are defined by polygons covering objects in the image. See Figure 2.11

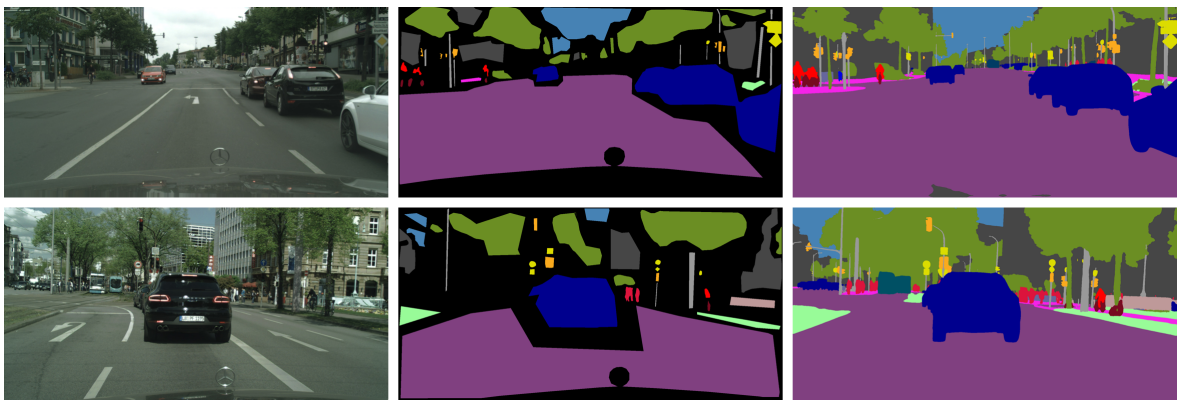


Figure 2.11: RGB image (left), Coarse annotation (center) and fine grained annotations (right) from the Cityscapes dataset [44]

Additionally, current synthetic datasets do not provide the needed variability to generalize to specific scenarios, such as the Semantic drone dataset[5].

When analyzing each dataset the discrepancies grow larger when looking at the semantic objects present in each. In Table 2.3 we include some datasets and the included labels.

[5]http://dronedataset.icg.tugraz.at/

Dataset										
Name	Synthetic	Open source	Point of view					Number of images	
			Wearable cameras		Fixed view		Free view	Coarse annotations	Fine annotations
			Ground view	Aerial view	Ground view	Aerial view			
Kitti	R	-	✓						200
WildDash	R	-	✓						4256
Cityscapes	R	-	✓					20000	5000
Mapilliary	R	-	✓		✓				25000
Semantic drone dataset	R	-		✓					400
VKITTI2	S	×	✓						21260
Synthia	S	×	✓		✓				220000
MSS	S	✓	✓	✓	✓	✓	✓		∞

Table 2.2: Comparison of different semantic segmentation datasets of urban scenes. Synthetic column indicates whether the dataset is real (R) or synthetic (S). On the open source column indicates whether the source code of the dataset is publicly available (✓) or not (×), for real dataset as there is no source code a - is assigned.

Name		Class																														
		Void			Flat					Construction						Object						Nature		Sky	Human	Vehicle						
	Real	Ground	Dynamic	Static	Road	Sidewalk	Parking	Rail track	Road marking	Building	Wall	Fence	Guard rail	Bridge	Tunnel	Pole	Pole group	Traffic sign	Traffic light	Billboard	Street light	Terrain	Vegetation	Sky	Rider	Person	Car	Truck	Bus	On rails	Bicycle	Motorcycle
Cityscapes	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Kitti	✓	✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
WildDash(rails)	✓	✓						✓				✓				✓										✓		✓				
WildDash	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mapilliary	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
VKitti		✓	✓		✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Synthia					✓	✓			✓	✓		✓				✓		✓	✓				✓	✓	✓	✓	✓				✓	✓
MSS					✓	✓			✓	✓	✓					✓			✓	✓			✓	✓		✓	✓		✓			

Table 2.3: Comparison of different semantic segmentation datasets of urban scenes. Each column corresponds to a semantic label, all semantic labels are grouped by general labels following the Cityscapes criteria.

Chapter 3

Generation of synthetic data for semantic segmentation

One of the key objetives of this project is analyzing the impact of video sequences generated with the *MSS* simulator for semantic segmentation. The goal of this chapter is to understand the different video sequences which can be obtained and how each generalize to real data. The final goal of this chapter is to generate a large dataset from the *MSS* simulator.

We start by creating a design criteria in order to generate a small sample set of frames categorized into 5 different categories: Pedestrian, Static camera, Helicopter, Car and Bus. After generating the sample set, we describe each video category and analyze the possible biases to understand the benefits of each category of video sequences. We follow by training models with each small set to obtain some performance metrics in order to have a valid metric for each of them. Finally, we conclude with a summary of the generated dataset and some conclusions on the experiments.

3.1 Description of the *MSS* sequences categories

We have divided the possible sequences generated by the simulator into 5 different categories. The nature of each of the proposed video categories, held intrinsic differences which may be interesting to consider:

- Pedestrian: A camera is attached to a walker with an egocentric point of view, due to this point of view, we find that pedestrians appear closer, therefore at a much bigger scale. Some biases are also present, the main surface which is present is a sidewalk, due to pedestrians walking over sidewalks instead of roads, furthermore this bias is different from the one present in Cityscapes, Mapilliary, and any real dataset for urban scenes, in the way that those present the center bottom part of all pictures being of class road and having to one side or both sides a sidewalk. This scenes present an overall bottom surface of sidewalk, and some examples of a road appearing on the margins of the image.

- Fixed camera: A camera is fixed in a desired position. This category aims at resembling video cameras on traffic poles. This kind of video category holds similar biases to the ones found in the car video category and real datasets for urban scenes such as Cityscapes, having an abundance of road and sidewalk instances in a similar geometrical disposition unlike the Walker category.

- Helicopter: A camera is attached to the underside of an helicopter with a fixed inclination to focus on the roads. This kind of category present similar to the walker category, we find unique biases to this category: Due to it's flying point of view, in some scenes buildings are captured from the sky therefore being the surface below the camera (appearing in the bottom half of the image). This makes this category centered on buildings and sky, understanding the surface shape of buildings.

- Car: A camera is attached to the driver with an egocentric point of view, a bias of having the front of the car always appearing at the bottom is optional, however this may be beneficial to it's generalization to datasets such as Mapilliary, where the same bias is persistent

- Bus: A camera is attached to the driver with an egocentric point of view. We find this possibility unique in the way that to the best of our knowledge there is no other dataset (Real or synthetic) where this point of view is provided.

In order to clearly classify each of the possible sequences we have designed the following classification by the point of view in Figure 3.1

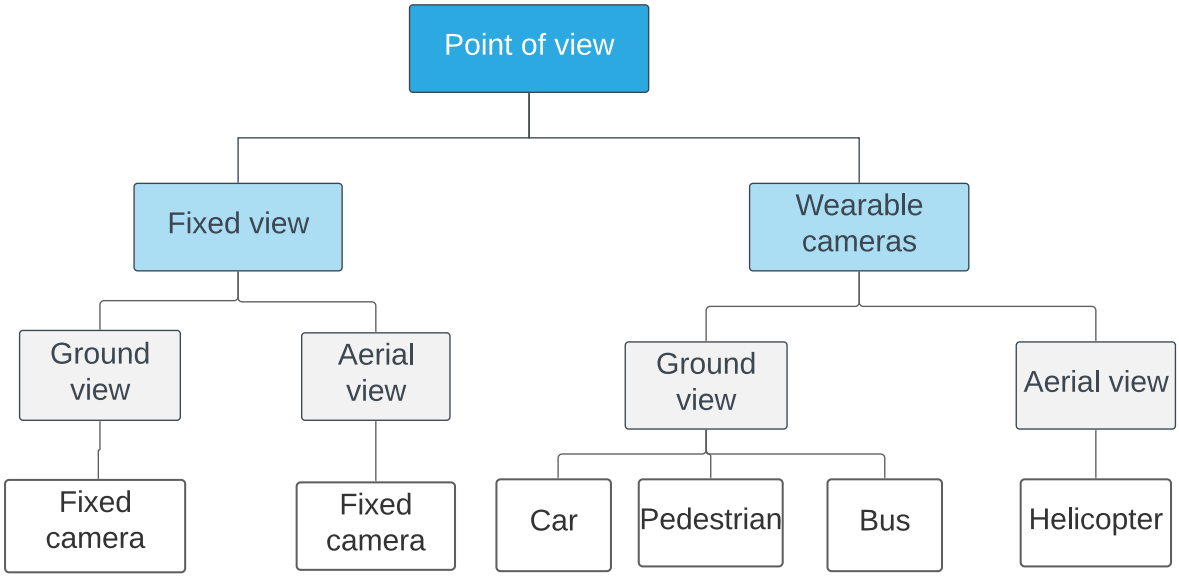


Figure 3.1: *MSS* sequences classification

In Figure 3.2 we include some sample frames from each sequence.

(3.2.1) Car sequence

(3.2.2) Helicopter sequence

(3.2.3) Pedestrian sequence

(3.2.4) Fixed camera sequence

(3.2.5) Fixed camera sequence

(3.2.6) Bus sequence

Figure 3.2: Samples of different synthetic images obtained with MSS

3.2 Design criteria

In this section we try to define a criterion of what is the best placement of cameras to obtain the best results as possible. Based on the results from [17] in object detection, we extrapolate to this problem that the more variability present on the training set the more the model needs to extrapolate to achieve good results. The criteria is divided into general aspects for all cameras and specific aspects for wearable and fixed cameras.

3.2.1 Wearable cameras

All cameras will be positioned in a way that the wearer is not visible in any frame. This is: having the camera at the margin of its surface. If the camera has an inclination then the camera will be placed at the margin point which leaves the most amount of surface of the car on the negative direction of the vector when separated by the hyper plane defined by the inclination vector and the margin point:

- S being the margin points of the wearable surface
- $H_{\{x,v\}}$ being the hyperplane defined by the inclination vector v and point x
- x such as that S$\bigcap H_{\{x,v\}^+}$ is maximal.

Figure 3.3 illustrates the positioning of the hyperplane with a car as an example.

Although on some real datasets such as Mapilliary, some frames present the bonnet front of the car, those scenes are marginal and in general urban scenes datasets tend to place the camera in a way that no part of the car is visible in the scene. See Figure 3.4 for a visual example

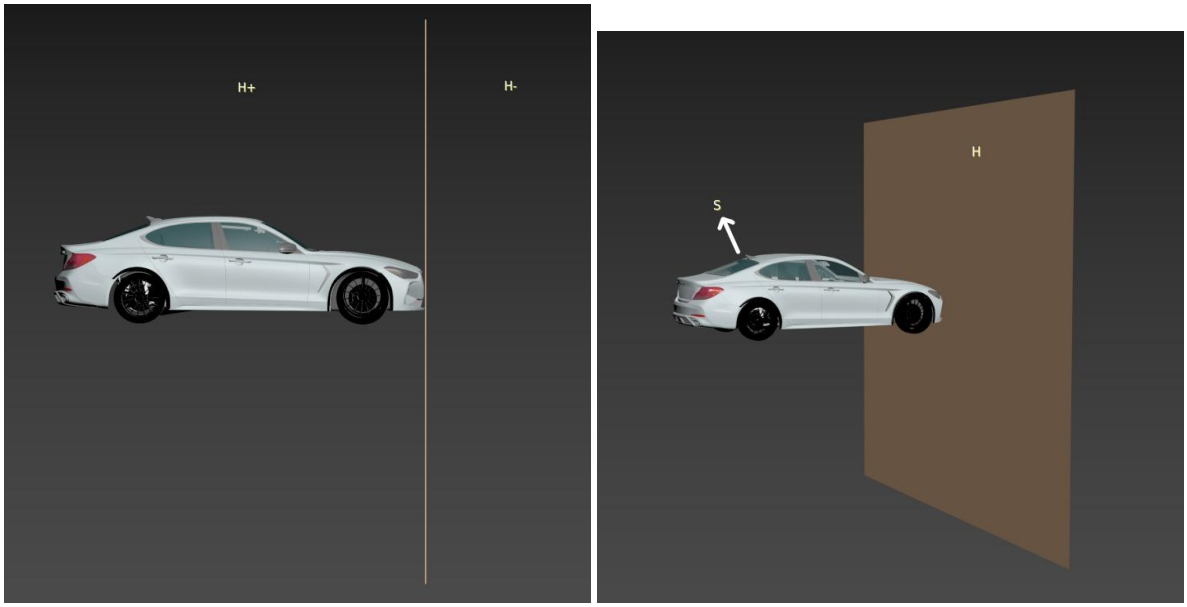


Figure 3.3: Illustration of camera positioning with a car representing the surface S.

Finally most wearable cameras will be placed on cars, having different sequences filmed from different cars. This criteria is motivated due to being more visually related to sequences found in real datasets.

Figure 3.4: Frame samples of wearable cameras obtained through the design criteria.

3.2.2 Fixed cameras

All cameras must be placed at a car front window's height or at a poles height. Parallel to the road for the ones placed at a car front window's height. For the ones place at a pole's height, an inclination so that the road is the main focus.

As intersections, roundabouts and turns are the least common scenarios when driving a car, which mainly present straight roads, will be the main focus in these sequences. This protocol will aims at alleviating biases from wearable cameras, were location of buildings, roads, sidewalks and sky are heavily biased.

3.2.3 General aspects

Due to the point of view of each of these sequences we can deviate a series of biases from each of the scenes, in Figure 3.5 we illustrate the global distribution of pixels per video category. We can see how each different category has a broader representation of each class, intuitively this should affect the final performance of the model when trained with said subcategory.

As discussed previously, each category of frames has certain biases due to the point of view, our goal is to take advantage of these biases in order to improve the performance of the model in specific scenarios where it may be under-performing.

3.3 Analyzing impact of the *MSS* sequences

In this section we discuss the potential benefits each category could provide to the final performance of the model when used in training. Due to the point of view of the scenes, each will yield some intrinsic benefits to the training, for example the scenes with a pedestrian egocentric point of view will increase the probability of the network of predicting the ground as a sidewalk, however due to the aspect ratio of the humans in this frames maybe will harm the performance of the network when predicting humans on a car point of view.

In order to empirically analyze the impact of each sequence, a *DeeplabV3* network is trained from scratch with these frames in order to understand how they affect to the final performance in general and in each specific class. These experiments aim at understanding the benefits of each video category, what potential benefits could be

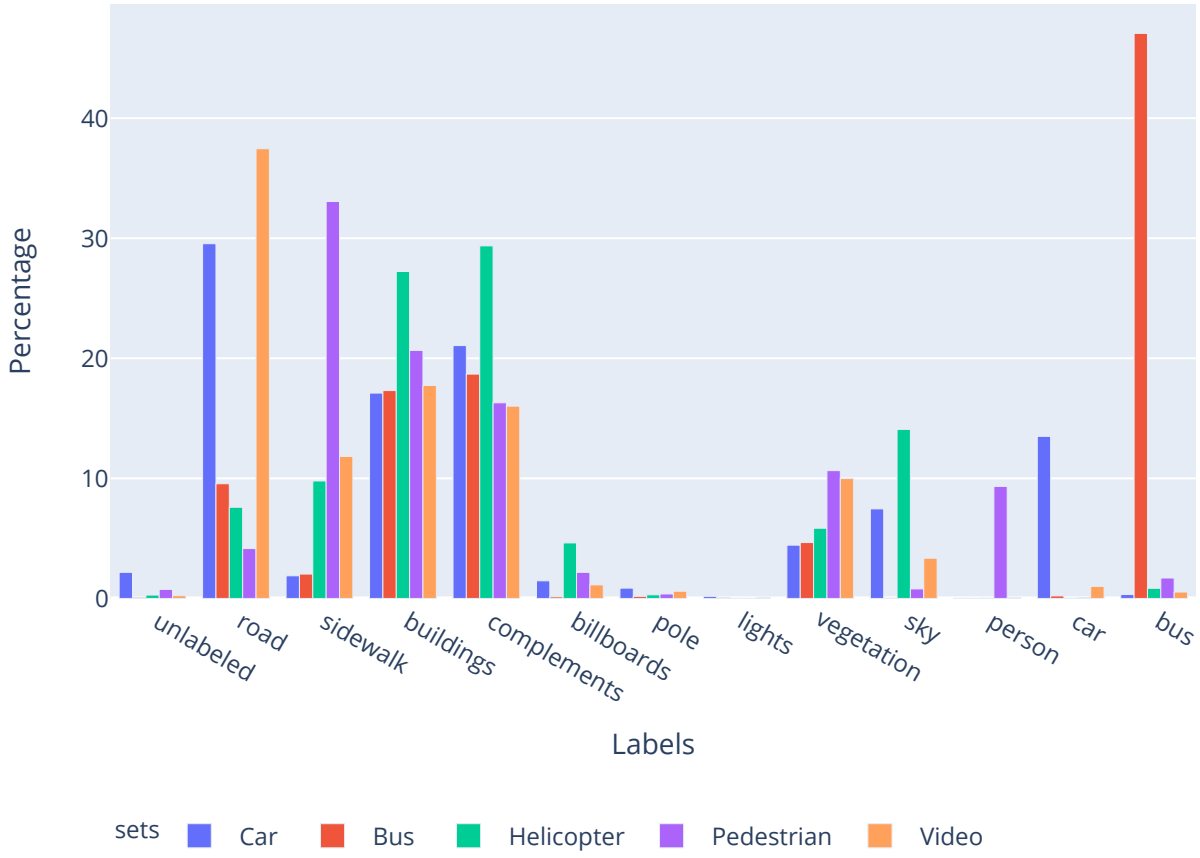


Figure 3.5: Distribution of labels per video category

extrapolated from using frames obtained by a persons egocentric point of view, or by a helicopters, etc.

Due to the flexibility of the *MSS* simulator we find that understanding the impact and it's possibilities we could make smarter choices when rendering the dataset. Furthermore, we could create a small subdataset to improve the performance of a model in some classes where it may be under-performing. Following the procedure of the previous experiment we divide the experiment into two analysis. Both experiments are extrapolated from [17], where they follow the same method in the object detection field.

- Mixture of real and synthetic data: This first approach serves as an initial exploration, what we are going to do is create different dataset corresponding of the whole synthetic dataset, being the *MSS* frames, and a proportion of the real dataset, 5%, 15% or 25%. With these new datasets we train a *deeplab* model and understand the results.

- Finetuning with a subset of the real data: With this experiment what we are going to understand is how are the learned weights on synthetic data generalizing to the real problem, in order to do so we are going to train the network with each of the synthetic datasets. With those learned models we finetune to the real data, but with only a proportion of the real data (5%, 15% or 25%).

3.3.1 Mixture of real and synthetic data

In this section two analysis are presented:

The per class accuracy comparison and the final performance measured with pixel accuracy and MIoU. These results are illustrated through tables presenting the obtained results. In Appendix B different graphs for the per class accuracy of the trained model on each of the real datasets test set, and graphs comparing the MIoU and MAP to the mixture of the one trained with real data (yellow star). are included. See Figures B.1 and B.2.

Results with hybrid datasets of synthetic data and *Cityscapes*

In Table 3.1 we include the performance divided by each of the classes represented. We would like to note how *Pedestrian* and *Fixed camera* seem to be the video sequences which provide a boost in performance.

We also would like to mention how in 3.1 when mixing *Cityscapes* and sequences, we see how biased sequences, such as *Pedestrian*, *Car* and *Bus* provide a bigger boost in performance to the biased class. This is, the best performance for the class car, is obtained with sequences filmed in a car's point of view, due to having more cars appearing. Same for bus and pedestrians.

In Tables 3.2 and 3.3 we present the global results for each synthetic and real mixture. We can see how including some synthetic data from sequences yields a boost in performance.

This performance boost can be seen in it's specificity, having a better performance in it's own test set, see Table 3.2, where the results of testing on the *Cityscapes* test set are presented. This results also indicate that *Pedestrian*, *Car* and *Fixed camera* seem to be the best inclusions out of the selected video sequences.

The boost can also be seen in it's abstraction potential, by having a better performance in other real test set. This is the case of Table 3.3, where the results of testing on the *Mapilliary* test set are presented. This results, following the previous trend, indicate once again that *Pedestrian*, *Car* and *Fixed camera* seem to be the best inclusions out of the selected video sequences.

Real set	Synthetic	Proportion	%Pixels per class											
			Unlabelled	Road	Sidewalk	Building	Billboard	Pole	Light	Vegetation	Sky	Pedestrian	Car	Bus
Cityscapes	-	1	0.00	0.87	0.32	0.59	0.17	**0.09**	**0.02**	0.67	0.58	0.26	0.62	0.21
Cityscapes	Fixed	0.25	0.00	0.9	**0.35**	**0.61**	0.17	0.08	0.00	0.70	**0.58**	0.26	0.63	0.18
Cityscapes	Pedestrian	0.25	0.00	**0.91**	0.33	0.61	**0.18**	0.05	0.00	0.70	0.57	**0.29**	0.65	0.18
Cityscapes	Helicopter	0.25	0.00	0.9	0.33	0.61	0.17	0.07	0.00	**0.72**	0.51	0.28	0.66	0.24
Cityscapes	Car	0.25	0.00	0.91	0.34	0.61	0.17	0.08	0.00	0.70	0.55	0.24	**0.67**	0.21
Cityscapes	Bus	0.25	0.00	0.91	0.34	0.61	0.2	0.09	0.00	0.70	0.57	0.26	0.63	**0.26**
Cityscapes	Fixed	0.15	0.00	0.91	0.33	0.6	0.15	0.06	0.00	0.70	0.56	0.23	0.61	0.16
Cityscapes	Pedestrian	0.15	0.00	0.9	0.31	0.6	0.14	0.05	0.00	0.69	0.53	0.28	0.62	0.21
Cityscapes	Helicopter	0.15	0.00	0.91	0.31	0.6	0.15	0.04	0.00	0.69	0.57	0.29	0.63	0.19
Cityscapes	Car	0.15	0.00	0.91	0.34	0.61	0.18	0.08	0.00	0.71	0.58	0.28	0.64	0.23
Cityscapes	Bus	0.15	0.00	0.91	0.32	0.6	0.17	0.09	0.00	0.71	0.52	0.25	0.64	0.17
Cityscapes	Fixed	0.05	0.00	0.91	0.31	0.58	0.12	0.05	0.00	0.68	0.54	0.24	0.61	0.14
Cityscapes	Pedestrian	0.05	0.00	0.91	0.31	0.57	0.14	0.05	0.00	0.68	0.52	0.22	0.61	0.11
Cityscapes	Helicopter	0.05	0.00	0.9	0.31	0.58	0.11	0.05	0.00	0.68	0.5	0.23	0.61	0.12
Cityscapes	Car	0.05	0.00	0.91	0.31	0.59	0.14	0.05	0.00	0.69	0.52	0.22	0.59	0.11
Cityscapes	Bus	0.05	0.00	0.9	0.27	0.58	0.11	0.04	0.00	0.69	0.54	0.21	0.61	0.09

Table 3.1: Per class results from mixing *Cityscapes* and synthetic data from the sample set. Tested on Cityscapes

Results with hybrid datasets of synthetic data and *Mapilliary*

Following the previous guides, we present the following three tables with the results of the *Mapilliary* test set.

In Table 3.4 we include the results of including synthetic data to a real dataset. We can see how in contrast to what happened in the *Cityscapes* scenario, the original full train set of *Mapilliary* produces better models than using just a portion with synthetic data. This can be explained by the fact that *Mapilliary* is 10 times bigger than *Cityscapes*. Therefore, bigger synthetic datasets must be used in order to challenge it's baseline performance.

However, in Table 3.4 we can see in the second row, which includes the results of mixing 25% of the original train set and the *Fixed camera* sequences, provides the best results compared with any other sequence.

Following in Tables 3.5 and 3.6 we include the global results of testing in the *Cityscapes* and *Mapilliary* test sets respectively. We can see how in this case, the best performances, only taking into account the scenarios where synthetic data was included, are obtained through using the *Fixed camera* and *Car* video sequences.

Proportion	Synthetic	MAP	MIoU
1	-	0.85	0.37
0.25	Fixed	0.87	**0.37**
0.25	Pedestrian	0.88	0.34
0.25	Helicopter	0.87	0.35
0.25	Car	**0.88**	0.37
0.25	Bus	0.88	0.35
0.15	Fixed	0.87	0.33
0.15	Pedestrian	0.87	0.33
0.15	Helicopter	0.88	0.34
0.15	Car	0.88	0.35
0.15	Bus	0.87	0.34
0.05	Fixed	0.86	0.32
0.05	Pedestrian	0.86	0.32
0.05	Helicopter	0.86	0.31
0.05	Car	0.86	0.32
0.05	Bus	0.86	0.31

Table 3.2: Global results from mixing *Cityscapes* and synthetic data from the sample set. Tested on Cityscapes

Proportion	Synthetic	MAP	MIoU
1	-	0.77	0.25
0.25	Fixed	0.78	0.25
0.25	Pedestrian	0.79	0.25
0.25	Helicopter	0.77	0.26
0.25	Car	**0.79**	**0.26**
0.25	Bus	0.77	0.25
0.15	Fixed	0.75	0.24
0.15	Pedestrian	0.76	0.24
0.15	Helicopter	0.75	0.24
0.15	Car	0.79	0.26
0.15	Bus	0.78	0.26
0.05	Fixed	0.74	0.23
0.05	Pedestrian	0.76	0.24
0.05	Helicopter	0.73	0.23
0.05	Car	0.77	0.25
0.05	Bus	0.77	0.24

Table 3.3: Global results from mixing *Cityscapes* and synthetic data from the sample set. Tested on *Mapilliary*

This comparison can be seen both in the *Cityscapes* and the *Mapilliary* test set performance.

In Appendix B we include visual representations of each of the experiments performance for an easier comparison. Figures B.2 include the results obtained when using *Mapilliary* dataset.

What we can extrapolate from these experiments is that when training with huge datasets such as *Mapilliary*, in order to over perform the results of the whole dataset marginal proportions such as 25% or less won't be enough.

In these experiments we find that *Fixed video*, *Car* and *Pedestrian* sequences tend to provide better results when a mixture of real and synthetic data strategy is used. When analyzing the classes benefits we find the following:

Using *Fixed video* sequences yield a better overall performance, granting a uniform performance where all categories tend to follow the same performance than the one with the whole real dataset. Therefore we believe that this sequences should be an staple in the *MSS* dataset.

Using the *Pedestrian* sequences provide improvements to the pedestrian performance, we believe that due to the differences in size and location forces the model to focus on generalities between both rather than biases of the real dataset, therefore improving generalization.

The *Car* sequences inclusion may help generalizing to other datasets, like in the *Cityscapes* train set in Table 3.3.

Removing the bias that the center part of the image is always a road and may be the front bonnet of the car. The appearance of that bias on the training set provided better results than any other sequence.

Finally both *Bus* and *Helicopter* sequences seem to have a benefitial impact in the classes where they have more representation, such as Bus for *Bus* and Vegetation and

Real set	Synthetic	Proportion	%Pixels per class											
			Unlabelled	Road	Sidewalk	Building	Billboard	Pole	Light	Vegetation	Sky	Pedestrian	Car	Bus
Mapilliary	-	1	0.00	**0.71**	**0.3**	**0.55**	**0.3**	**0.16**	**0.05**	**0.71**	**0.9**	**0.2**	**0.62**	**0.17**
Mapilliary	Fixed	0.25	0.00	0.70	0.27	0.53	0.25	0.14	0.02	0.69	0.89	0.17	0.58	0.14
Mapilliary	Pedestrian	0.25	0.00	0.69	0.26	0.5	0.25	0.13	0.02	0.65	0.89	0.15	0.58	0.13
Mapilliary	Helicopter	0.25	0.00	0.69	0.26	0.52	0.27	0.14	0.02	0.68	0.89	0.15	0.58	0.13
Mapilliary	Car	0.25	0.00	0.70	0.26	0.52	0.26	0.13	0.02	0.68	0.89	0.14	0.58	0.15
Mapilliary	Bus	0.25	0.00	0.69	0.22	0.51	0.23	0.12	0.03	0.68	0.9	0.16	0.56	0.10
Mapilliary	Fixed	0.15	0.00	0.65	0.18	0.47	0.18	0.06	0.00	0.61	0.87	0.12	0.55	0.13
Mapilliary	Pedestrian	0.15	0.00	0.69	0.24	0.52	0.25	0.10	0.00	0.67	0.88	0.17	0.58	0.12
Mapilliary	Helicopter	0.15	0.00	0.69	0.22	0.51	0.25	0.13	0.00	0.67	0.89	0.16	0.57	0.12
Mapilliary	Car	0.15	0.00	0.69	0.23	0.51	0.24	0.13	0.00	0.68	0.89	0.15	0.57	0.15
Mapilliary	Bus	0.15	0.00	0.69	0.25	0.52	0.25	0.13	0.00	0.68	0.89	0.16	0.58	0.12
Mapilliary	Fixed	0.05	0.00	0.66	0.2	0.48	0.21	0.13	0.00	0.65	0.88	0.14	0.54	0.09
Mapilliary	Pedestrian	0.05	0.00	0.66	0.19	0.48	0.19	0.10	0.00	0.65	0.88	0.14	0.54	0.10
Mapilliary	Helicopter	0.05	0.00	0.67	0.15	0.48	0.22	0.13	0.00	0.64	0.86	0.12	0.54	0.10
Mapilliary	Car	0.05	0.00	0.66	0.19	0.49	0.23	0.13	0.00	0.66	0.88	0.13	0.55	0.13
Mapilliary	Bus	0.05	0.00	0.66	0.21	0.48	0.22	0.12	0.00	0.64	0.87	0.14	0.55	0.09

Table 3.4: Per class results from mixing *Mapilliary* and synthetic data from the sample set. Tested on Mapilliary

Proportion	Synthetic	MAP	MIoU
1	-	**0.85**	**0.36**
0.25	Fixed	0.85	0.34
0.25	Pedestrian	0.84	0.34
0.25	Helicopter	0.85	0.34
0.25	Car	0.83	0.33
0.25	Bus	0.85	0.34
0.15	Fixed	0.75	0.25
0.15	Pedestrian	0.85	0.31
0.15	Helicopter	0.85	0.33
0.15	Car	0.84	0.33
0.15	Bus	0.84	0.34
0.05	Fixed	0.84	0.30
0.05	Pedestrian	0.83	0.29
0.05	Helicopter	0.82	0.30
0.05	Car	0.82	0.29
0.05	Bus	0.84	0.30

Table 3.5: Global results from mixing *Mapilliary* and synthetic data from the sample set. Tested on Cityscapes

Proportion	Synthetic	MAP	MIoU
1	-	**0.87**	**0.39**
0.25	Fixed	0.86	0.36
0.25	Pedestrian	0.85	0.35
0.25	Helicopter	0.85	0.33
0.25	Car	0.85	0.36
0.25	Bus	0.85	0.35
0.15	Fixed	0.83	0.29
0.15	Pedestrian	0.85	0.33
0.15	Helicopter	0.85	0.32
0.15	Car	0.85	0.33
0.15	Bus	0.85	0.33
0.05	Fixed	0.84	0.31
0.05	Pedestrian	0.84	0.30
0.05	Helicopter	0.83	0.30
0.05	Car	0.84	0.34
0.05	Bus	0.83	0.31

Table 3.6: Global results from mixing *Mapilliary* and synthetic data from the sample set. Tested on Mapilliary

Billboard for *Helicopter* sequences. However we can see how in the classes where they have little to no representation or there is a drastic shift in point of view, such as Billboard and Sidewalk for *Bus* reduces the performance of the model. We believe that this is due to the greater gap among those sequences and real images, whether is for the point of view or angle of representation.

3.3.2 Finetuning with real data pretrained models with synthetic data

For this experiment we compare the evolution of the training when using pretrained weights on synthetic scenes. In this experiment our goal is to find which video categories when used as training data to a neural network are more prone to generate better weights for the network to generalize to real data. In a way, the better the weights generates implies that richer features had been extracted, therefore providing a better basis to use as initial weight of future challenges or tasks.

Following the ideas from [45], where they analyze the usage of self supervised learning in Computer Vision tasks and measure the performance when finetuning with pretrained weights on pretext tasks and random weights.

Following their experimental protocol of training with a set learning rate the pretext task, in our case using synthetic data, and following with a smaller learning rate on the goal task, in our case using real data.

The experiment is inspired from [17], where they present the idea that mixed training should not be the chosen approach due to the randomness and difference in size of datasets. Which ends up leading to models which present a great performance for synthetic data but lack abstraction power to translate to real scenarios. In order to ensure the conclusions gathered from the previous experiments we induce this experiment to prove it's robustness.

Figures 3.63.73.8 illustrates the evolution of the training curves when using different proportions of real data on pretrained *DeeplabV3* models. The metrics are obtained by measuring the performance on the test set of the real set used in the mix.

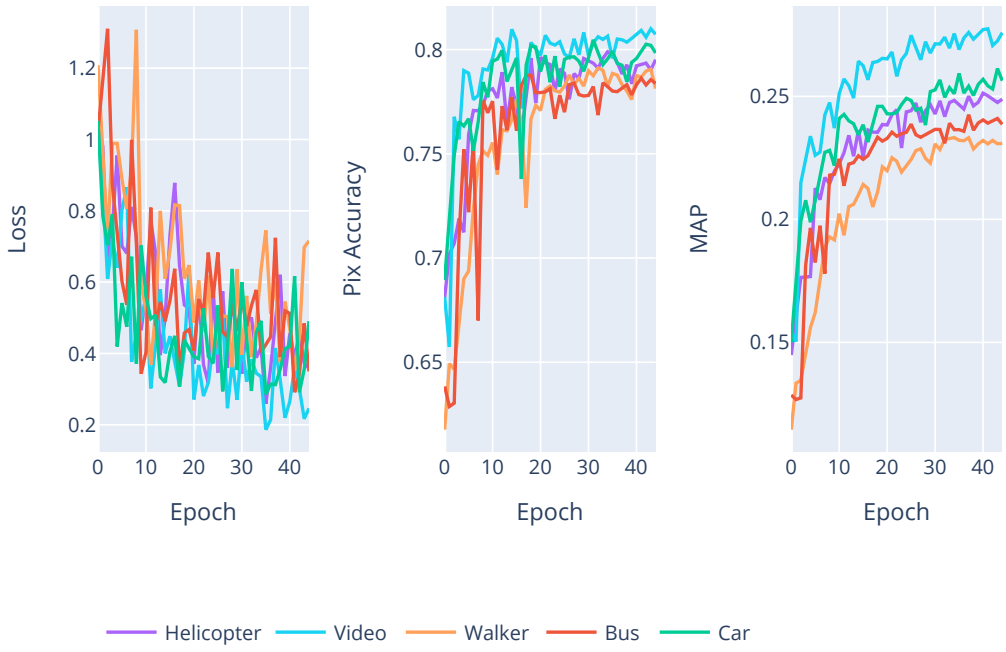
We can see how *Fixed camera* seem to consistently produce better results, specially joint with a portion of the *Cityscapes* train set and tested on the respective test set.

When looking a the results with *Mapilliary*, *Fixed camera* seem to provide better initial weights. However, when a 25% of the *Mapilliary* train set is used the *Car* and *Helicopter* sequences seem to present some competition to the *Fixed camera* sequences.

From this experiment we can assume that *Fixed camera* sequences seem to be a great asset from the *MSS* simulator, as was presented in the previous experiment.

3.4 Dataset generation based on the *MSS* simulator

After analyzing the design criteria for the video sequences, we follow by explaining the methodology used to generate the sequences and the ideas behind it. Starting with a brief tutorial on how to obtain the video sequences, going through the variations of the sequences and finally giving a brief overview of the whole dataset.

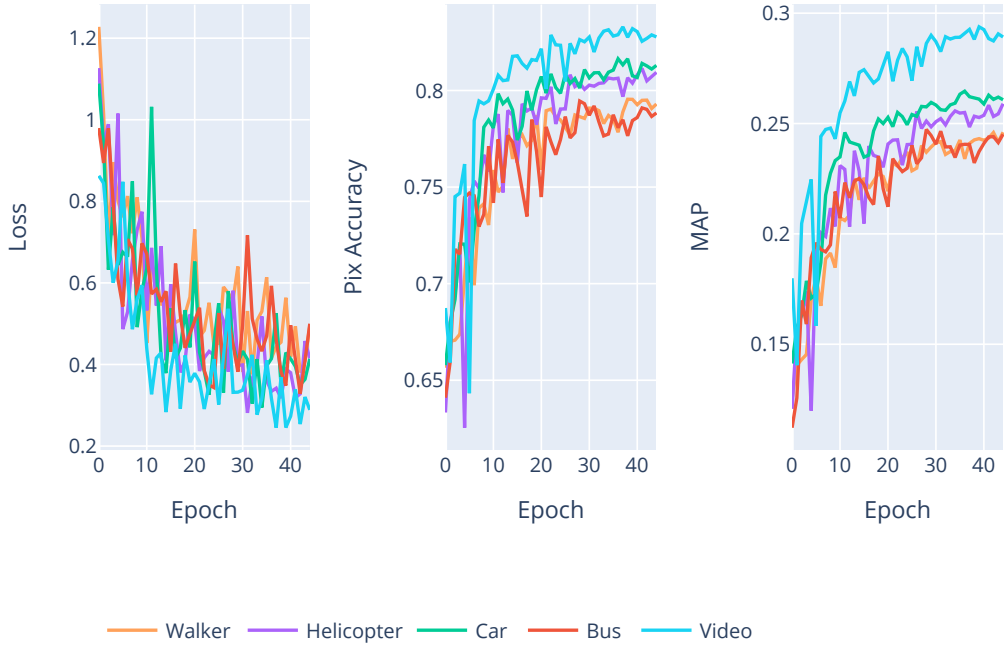


(3.6.1) Finetuning with *Mapilliary* and testing on *Mapilliary*

(3.6.2) Finetuning with *Cityscapes* and testing on *Cityscapes*

Figure 3.6: Finetuning pretrained models on each video sequence with a 5% of real data. Tested on the real data test set.

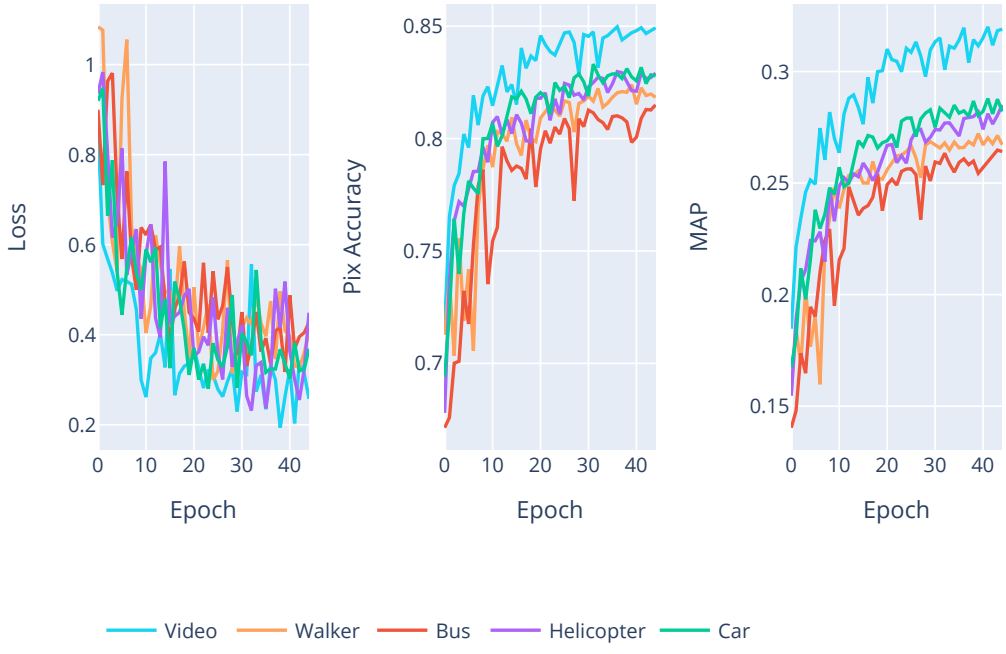
(3.7.1) Finetuning with *Mapilliary* and testing on *Mapilliary*



(3.7.2) Finetuning with *Cityscapes* and testing on *Cityscapes*

Figure 3.7: Finetuning pretrained models on each video sequence with a 15% of real data. Tested on the real data test set.

(3.8.1) Finetuning with *Mapilliary* and testing on *Mapilliary*



(3.8.2) Finetuning with *Cityscapes* and testing on *Cityscapes*

Figure 3.8: Finetuning pretrained models on each video sequence with a 25% of real data. Tested on the real data test set.

3.4.1 Scene preparation

The *MSS* simulator dynamically generates pedestrians in the park and cars through the road, however buses are not included, neither pedestrians throughout the city. Therefore, we need to manually position the buses we are interested in positioning and some pedestrian throughout the city.

To do so we need to:

1. Create an empty folder for the objects to be included. To do so click right button and select Create Empty. See Figure 3.9.1

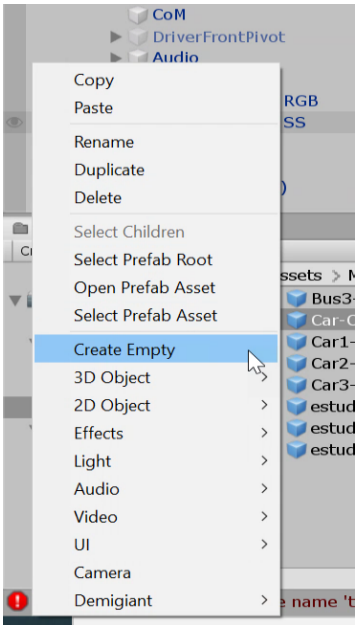
2. Open the Prefabs folder, it is located at Assets/MapaSemantico/Prefabs. See Figure 3.9.2

3. Select the desired object, the predefined objects are: Bus, Car or pedestrian (estud3). Include the element into the simulation by dragging and droping the element into the created empty object in step 1

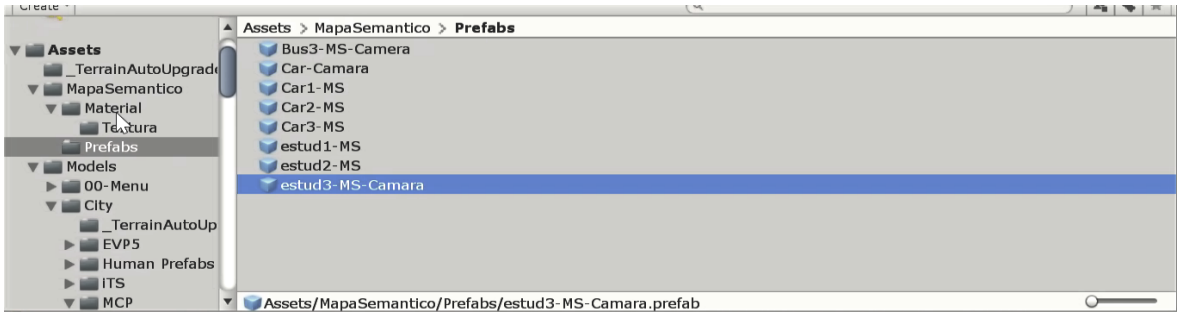
4. Deselect the CameraMovil RGB and CameraMovil Sem in order to speed up the simulator, although it is not mandatory it is highly recommended. Those are located inside the object in the folder RefGiro. See Figure 3.9.4

5. Duplicate them as many times as needed by right button on the object, select duplicate. See Figure 3.9.3

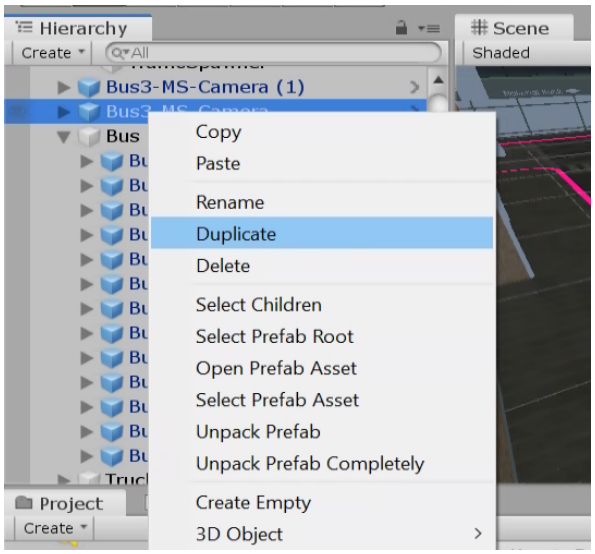
6. Place them as you want as any other object in the scene. You can use coordinates or the predefined motions of unity.



(3.9.1) Create an empty folder in unity



(3.9.2) Semantic object prefabs location



(3.9.3) Duplicate an object

(3.9.4) CameraMovil RGB and CameraMovil Sem location in prefabs objects

Figure 3.9: Positioning new predefined objects tutorial

3.4.2 Adding new objects to the semantic layer

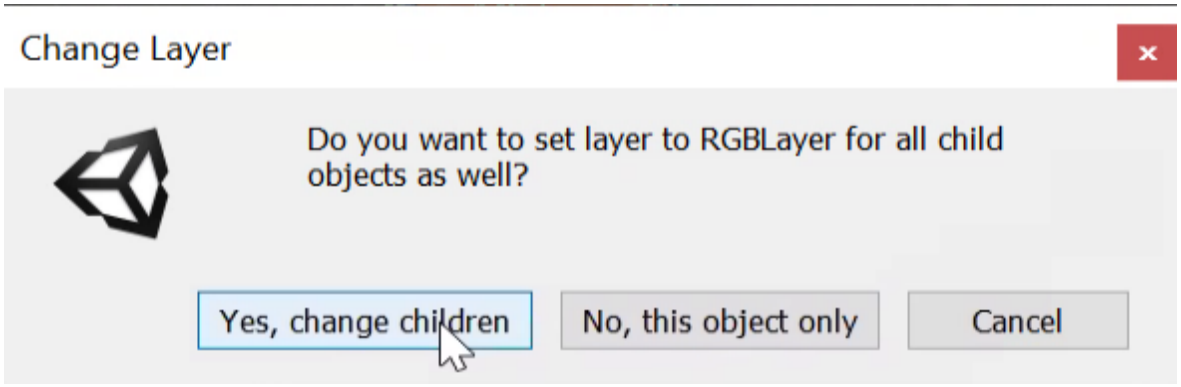
Given that the *MSS* simulator provide us with different objects we can make usage of them. However, not all of them are semantically labeled, therefore we need to do the following to introduce a new object to the semantic layer.

1. Include the prefab item into the simulator, for example we choose a new car model, see Figure 3.10.1
2. Duplicate all the contents of the prefab by right click on the mouse duplicate.
3. Change the original elements to be placed into the RGBlayer, see Figure 3.10.2 and select yes in the following popup, see Figure 3.10.3.
4. Repeat the process with the duplicated elements but on the Semanticlayer.
5. Drag and drop the label texture of the object, in our example car, from the AssetsMapa Semanticotexturas folder, to each of the duplicated components, see Figure 3.10.4

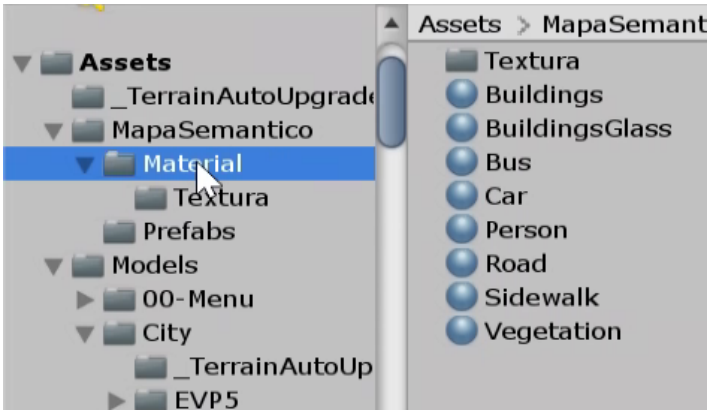
If everything was done correctly the end result should be similar to the one in Figure 3.11

(3.10.1) Adding a prefab object

(3.10.2) Setting the RGB layer elements



(3.10.3) Accept message from unity



(3.10.4) Semantic textures

Figure 3.10: Labelling new objects tutorial

(3.11.1) RGB Layer (3.11.2) Semantic Layer

Figure 3.11: Visual results of a new object in the *MSS* simulator

3.4.3 Camera setting on a static position

In order to set a camera on the *MSS* environment and obtain the captured images one have to:

1. Open prefabs: In this folder you will find the preset cameras with all functionality needed already positioned. This folder is located at the bottom left corner of the unity interface, see Figure 3.12.1

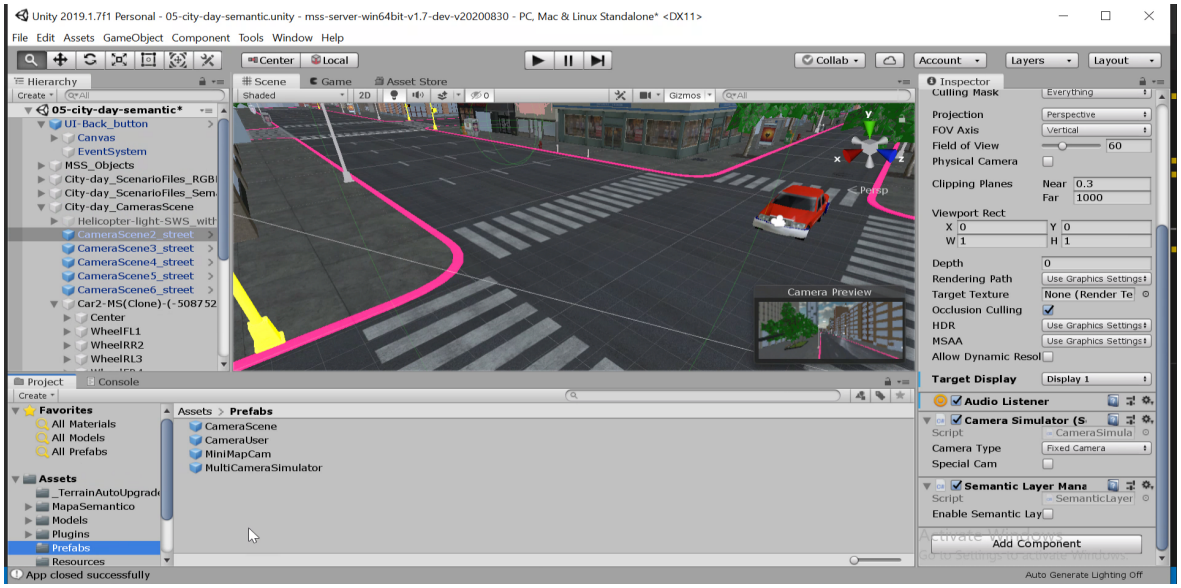
2. Setting a camera: In order to set the camera you just have to drag and drop the CameraScene label to the City-day_CamerasScene folder, see Figure 3.12.2

3. Change name and position of the camera: In order to identify each camera change the name by change the name on the top left corner.

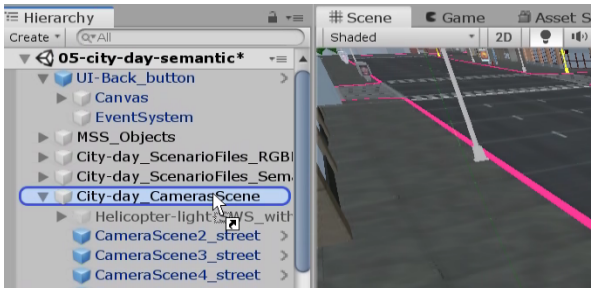
4. Setting camera to capture semantic segmentation information: In order to capture the ground truth of the sequence another camera must be placed at the same position with the feature of semantic segmentation enabled. The feature is activated through a checkbox at the bottom of the inspector menu (with the desired camera is selected) with the name of Semantic Layer Manager, see Figure 3.12.3

5. Starting the simulation: To start the simulation and capture videos we have to press the play icon, a pop up will appear with the logo start simulator, see Figure 3.12.4, press it.

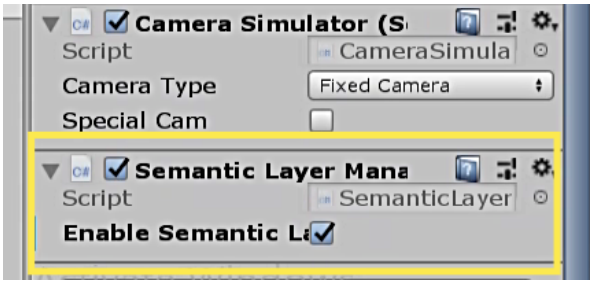
6. Record video: In a terminal execute the script record_selected.py. The program will record until the user press the key "Esc".



(3.12.1) Camera location in *MSS* simulator



(3.12.2) Setting a camera on the *MSS* virtual city



(3.12.3) Semantic layer *MSS* enabled

(3.12.4) Start *MSS* simulator.

Figure 3.12: (a) Camera location in *MSS* simulator. (b) Setting a camera on the *MSS* virtual city, inorder to maintain cohesion and make usage ofexisting scripts to ease user manipulation weencourage the location of all cameras in theCity-dayCamerasScene folder. (c) Enable camera to cap-ture ground truth informa-tion. (d) Start *MSS* simulator.

3.4.4 Wearable camera setting

The previous tutorial covered the basics on how to set a camera and record it, however the *MSS* simulator provides a feature of Wearable cameras, where the cameras are attached to an animated object and moves with the object.

1. Manually generated items, Select the object to place the camera. In order to illustrate this process a video from [10] is available in vimeo

*1 Dynamically generated items. Use one of the auto-generated objects by the simulator. In order to do so we have to:

   1.1. Start the simulator, see Figure 3.12.4

   1.2. Copy one of the auto-generated elements. When the simulator is running a folder is automatically generated, TrafficCarsContainer, this folder is the location where the simulator dynamically places the generated cars and pedestrians. Choose one and copy it. To copy an element just press with the right button of the mouse the desired object and select copy, ("Ctr+C" works as well).

   1.3. Place it in the City-day_CamerasScene folder. To do so press with the right button of the mouse on the folder and select paste ("Ctr+V" works as well).

2. Open prefabs like previously explained.

3. Select CameraScene and place it into the selected object, same process as before with a drag and drop motion. This will make the position and angle of cameras be relative to the object.

4. Make sure both cameras are inside the object and with the same exact position. As a guideline we will use for cars position X=0, Y=1, Z=3, and all angles set to 0. For pedestrians we will use position X=0, Y=2.5, Z=0.5, and all angles set to 0.

5. Starting the simulation as previously explained.

6. Record video as previously explained.

3.4.5 Dataset generation strategies

After providing all the information needed to generate the dataset we provide some insight on how we generated the *MSS* dataset used in the following experiments:

As the *MSS* provides a tool to change the amount of vehicles are generated we use this feature to our advantage. In order to discriminate each video sequence a folder scheme is used, see Figure 3.13 For each video category 5 subfolders can be found, 33, 50, 100, 300, 500 cars, in each of them 2 subfolders can be found, RGB containing the RGB videos, Semantic containing the GT videos and a file named Train.csv which includes all the relative paths to each video of said category and amount of cars. Besides, a MSS_total.csv can be found which includes all relative paths to all videos.

This structure is selected in order to allow future usage of the dataset with a curriculum learning strategy if desired, where the complexity is increased by increasing the number of cars in the city.

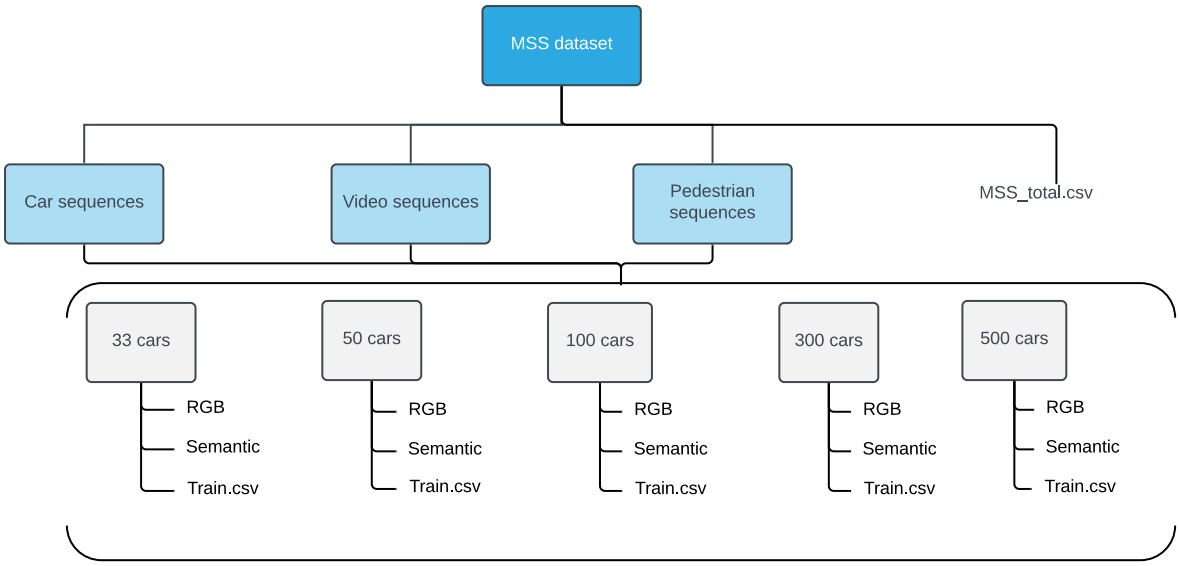


Figure 3.13: Folder scheme of the *MSS* dataset

3.5 Summary

In this chapter we analyzed the different video sequences which can be obtained from the simulator and the biases which each of them presented.

As can be seen throughout literature on the topic [[17], [18],[9],[19], [46]] the power of synthetic data comes from the variability which provides, forcing the network to focus on general aspects of the instances rather than colors and biases.

The *MSS* simulator provide us with two different type of cameras, wearable cameras and fixed cameras. Both of these cameras have their pros and cons. While wearable cameras provide more diversity to the frames due to changing the scenario, there are some instances which are less common, such as turns, roundabouts and intersections. Therefore, we can film those scenes with fixed cameras. This creates a dataset which has the diversity of a real dataset. However, making greater emphasis on uncommon scenarios, in an attempt to remove biases. Such as the sidewalks appearing always on both sides of the picture with a road in the middle, and so on.

Through a first inspection we find that whether we use synthetic data for a joint training or as initial weights for fine tuning, certain characteristics improve the final result.

Similarities between synthetic and real data: Although there is always a persistent domain gap, when synthetic data is used to gather initial weights followed by a fine tuning in real data. We find that similar scenes from fixed cameras and cars have a better performance when used for fine tuning, see Figures **??**.

Smaller objects tend to be punished harder when joint trained with synthetic data, see Table 3.1. Objects such as Pedestrians, Lights and Poles tend to have the worst performance. The inclusion of sequences from a pedestrian egocentric point of view seem to help the network to detect pedestrians.

Finally, in order to summarize the generated dataset, we provide the following Table 3.7 comparing our dataset with the some of the most popular datasets in semantic segmentation. Due to the generation strategy, we can see how our dataset includes

more amount of buses and sidewalk pixels proportionally than the other datasets. In comparison with the *Synthia* dataset, another synthetic dataset, we believe that ours holds a tighter similarity to the *Mapilliary* dataset and the *Synthia* to the *Cityscapes* dataset.

Dataset	Image size	Frames	Proportion of pixels per class											
			Unlabelled	Road	Sidewalk	Building	Billboard	Pole	Light	Vegetation	Sky	Pedestrian	Car	Bus
MSS	480×640	89363	0,00	0,34	0,07	0,29	0,02	0,00	0,00	0,07	0,10	0,00	0,05	0,03
Synthia	480×640	220000	0,02	0,29	0,03	0,20	0,00	0,01	0,00	0,21	0,13	0,00	0,09	0,01
Mapilliary	1024×2048	25000	0,04	0,16	0,03	0,16	0,02	0,02	0,00	0,19	0,31	0,01	0,05	0,02
Cityscapes	1024×2048	5000	0,04	0,49	0,03	0,16	0,01	0,01	0,00	0,14	0,02	0,02	0,08	0,01
Kitty	480×640	500	0,02	0,36	0,02	0,06	0,00	0,01	0,00	0,32	0,11	0,00	0,09	0,01

Table 3.7: Summary table comparing the generated dataset with current state of the art datasets.

Chapter 4

Evaluation Methodology

In this chapter our goal is to summarize the evaluation methodology which we have followed during the project. We start by addressing the Testing environment we've chosen, including the visualization tools used and the programming design used to full fill the task. Continuing with the datasets we've included in our study and a study of each of them. We follow by addressing the performance evaluation metrics we've chosen for the problem and defining them. We conclude with the evaluation protocol, giving a brief introduction to our goals with each experiment and the insights we want to gather.

4.1 Testing environment

For this project we chose a *Deeplabv3* architecture to perform our experiments. Our goal is to reuse and to generalize the code as much as possible. In this section we detail the software paradigm and the visualization tools used.

4.1.1 Software paradigm

For this project we've used Python 3.7 [47] for our implementation with Pytorch [48] as our deep learning framework. Our goal was for the structure of the code to be as general as possible, in order to achieve this we followed a OOP (Object Oriented Programming) approach. Two key classes are proposed for the project (see Figure 4.1):

- Training: The core class, is in charge of the training of each of the networks with the desired training data, periodically validates data with a selected validation data and at the end of the training generates some sample images of the input output and gt of the model. Manages to make the user invisible to the differences between each of the available networks.

- Loader: In charge of fetching the data, data augmentation and mixing the specified proportions of each of the desired datasets, takes as input a filename of the dataset csv to be used or a dictionary with the csv file and the proportion of that dataset normalized in the $0-1$ range. As there is a preprocessing where a csv for each of the training set is available we can generate dynamically different random sets with the desired proportions each loop, in order to ensure replicability a random seed is set to 0, however it could be changed so every training sees a different training set.

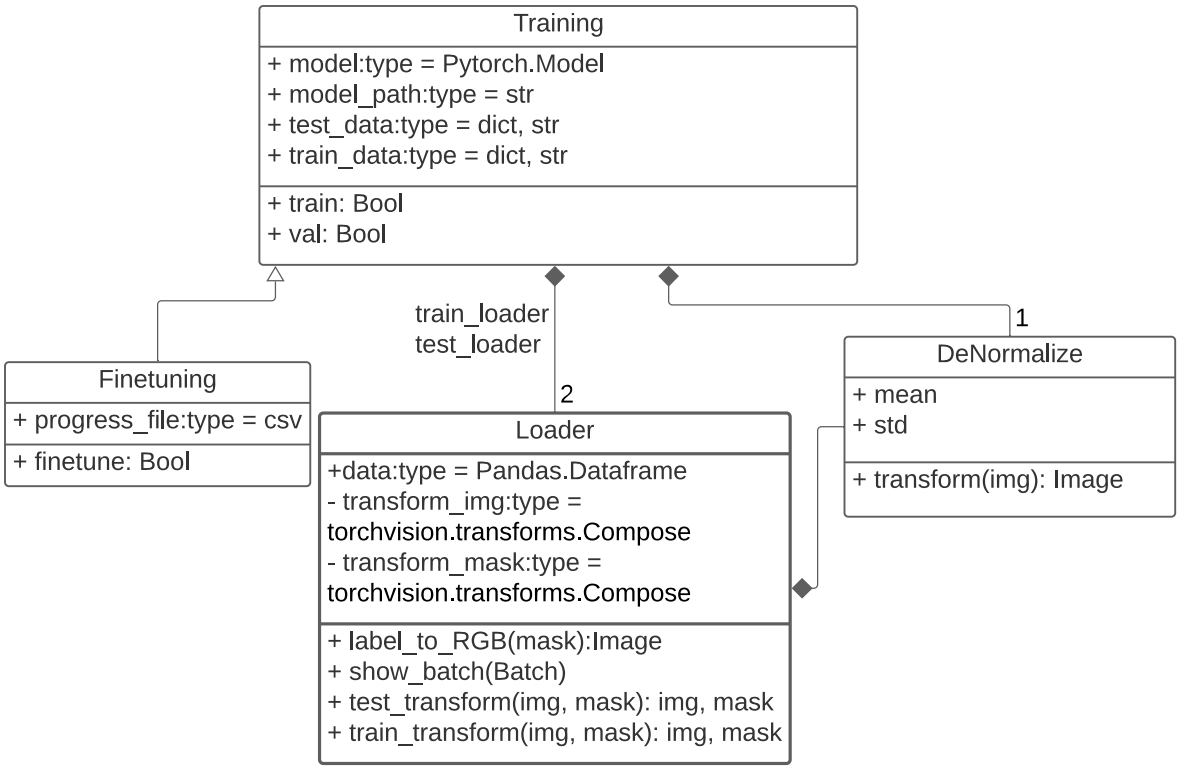


Figure 4.1: Class diagram

As we were going to do exhaustive training in order to speed the process we stored the labelled images into a numpy format in order to save the preprocessing of the labels. The preprocessing consist of the following two steps:

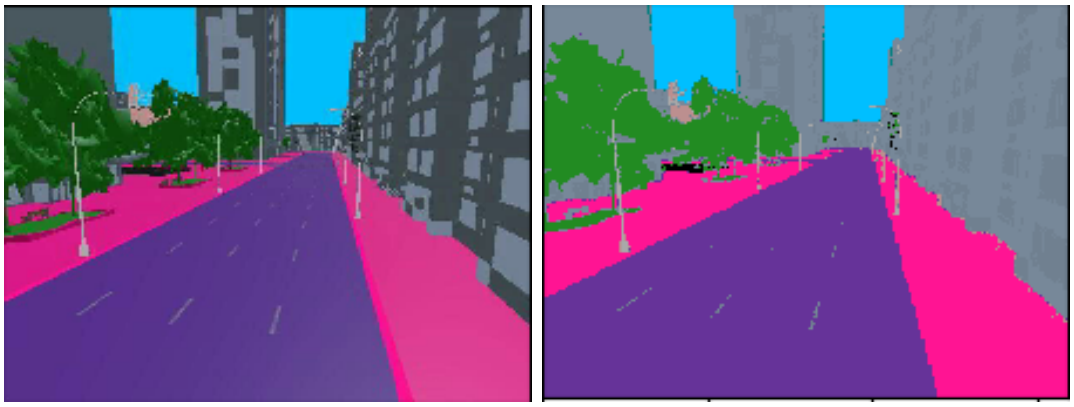
First, normalize each of the ground truth maps. The MSS tool creates a layer over each object of a solid color, however due to compressing format and light conditions of the city, different shades of the color may be present in the final GT map, therefore we need to correct those imperfections. The corrections are performed automatically, through a nearest centroid classifier based on euclidean distance. Different clusters are generated, based on proximity the script assigns to a label. Then, a median kernel is applied to remove small spots of light which may change the GT label. See Figure 4.2 for a visual example.

Second, generate the train test and validation split. A script to generate a partition was created. It goes through the files assigning a random set of images to each of the splits given the percentages as input. It generates 3 csv files for each of the sets with the path to the image and the label mask.

As a summary, the process of running an experiment is:

1. Downloading the data: Download the desired dataset to be used in the experiment. Usually most datasets have a csv indicating which color represent each class, if this csv does not exist, manually one has to create it or insert it into a dictionary for the next steps to work.

2. Generating the train, validation and test csv: In order to generalize as much as possible a csv is generated for all datasets. This is done through the script



(4.2.1) Output from the MSS simulator. (4.2.2) Corrected GT.

Figure 4.2: MSS simulator ground truth correction.

csv_create, which for each GT map generates a numpy array, which is stored to speed up the fetching process while training, and generates a csv for test, validation and training, containing the gt path and the image path. This allows the dataloader to simply load an image per row on the csv file.

3. Performing the experiments. Class Training and it's heritage Finetuning contain the functionality to perform training or finetuning with a given the input size of the images and a mixture of proportions of selected datasets, one only needs to select proportion of 1 to choose a complete dataset. Then it proceeds to do training. The protocol is always to save model whenever the test performance was increased.

In Appendix D we include in Figure D.1 the Sequence diagram of the experiments.

4.1.2 Visualization Tools

In order to ease the process of visualization and to have all the experimental results into one integrated application we have created a visualization framework based on Dash and Plotly[1].

1. Dash. Dash is a Python framework for building web analytic applications written on top of Flask [49], Plotly.js, and React.js.

   Dash works as the basis of the module providing a web infrastructure and containers for each of the plotly graphs.

2. Plotly. The plotly Python library is an interactive, open-source plotting library. Creating reactive and live editable plots. Due to the manageability it provides, of being able to zoom to different aspects and remove dynamically elements of the plot we believed that this tool was best suited for the goal of this project of understanding and extrapolating learning protocols based on performance metrics.

As the main goal of this project is to gather knowledge the framework is designed to be able to filter and manipulate the represented data as much as needed.

The visualization framework implementation is divided into two different aspects:

[1] https://plot.ly

Data manipulation.

The visualization framework needs to be feeded with data gathered through the experiments, the implementation followed a direct scheme, where the experiments outputs are in a csv format. The csv contains a header in order to name each axis and from the second row onward the results obtained. Two different categories of results are differenciated:

Evolution results. Where the advance of the training was the main focus. This results have as the first five colums the epoch number, the test dataset used, the training loss, the mean average presition and the MIoU. For each of this results $n \times 3$ figures (n being the number of test sets used) are plotted. One for each of the metrics against the epochs.

Final results. This experiments measure the final results obtained for the test sets of the network. This results have as the first four components the train and test sets used the MAP and the MIoU and then followed by the AP of each of the classes used. For each of this generated files, $n \times 2$ figures (n being the number of test sets used) are plotted. One scatter plot for the MAP against the MIoU and one bar plot for each of the classes AP.

Each of this figures is obtained through Pandas Dataframes [50] manipulation and Plotly.

Design implementation.

In order to access easily to each of the figures generated the server is divided into different tabs, one per experiment.

Inside of each of the tabs two drop-down menus are available, each of the drop-down menus for the train and test sets used, this drop downs filter the results to be plotted. In addition, each element in the legend can be clicked in order to remove those results of the graph and further clean the target data, a screen shot is available in Figure 4.3.

This implementation is done through Dash HTML components.

As a summary see Figure 4.4.

As it is a web service, it has the potential to be implemented in a server and have different collaborators uploading their csv files with their results and have a dynamic resource to share and visualize results among peers.

4.2 Performance evaluation metrics

The following metrics and algorithms are used to evaluate the performance of the experiments.

4.2.1 Metrics

Mean Intersection over Union

The Jaccard index,developed by Paul Jaccard, is a statistic used for gauging the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets. In computer vision is widely used to measure the accuracy of

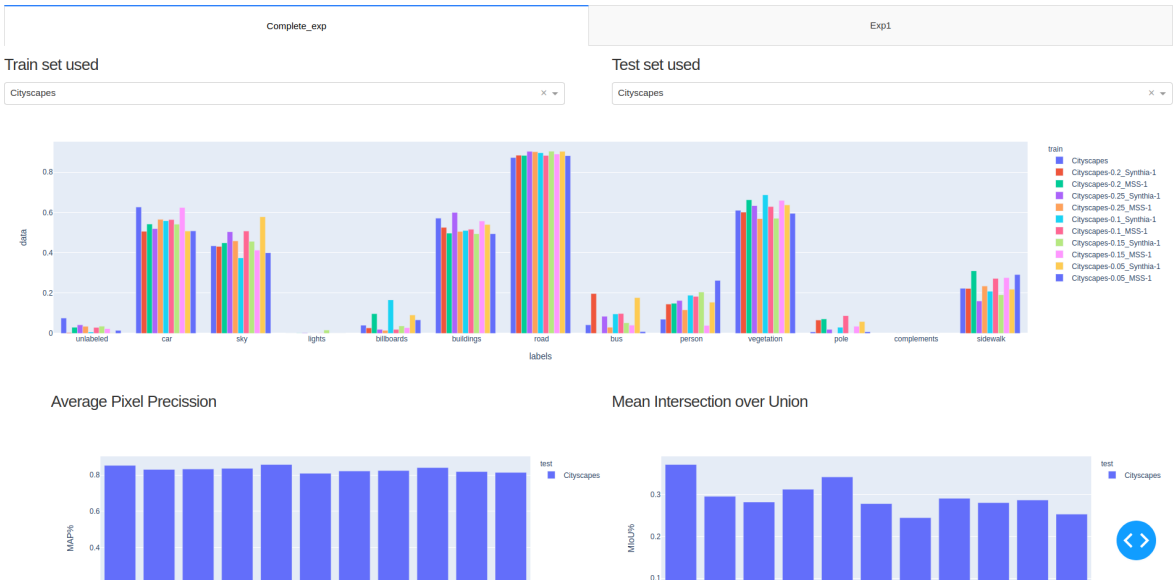


Figure 4.3: Screen capture of the visualization framework

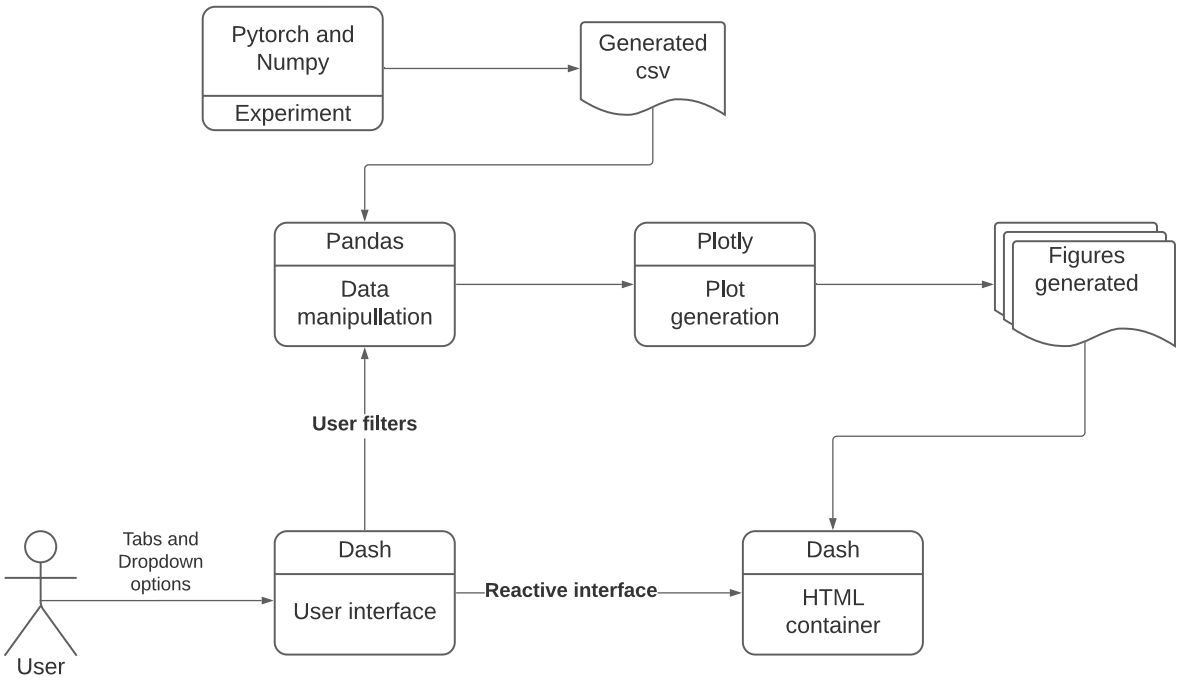


Figure 4.4: Interface diagram

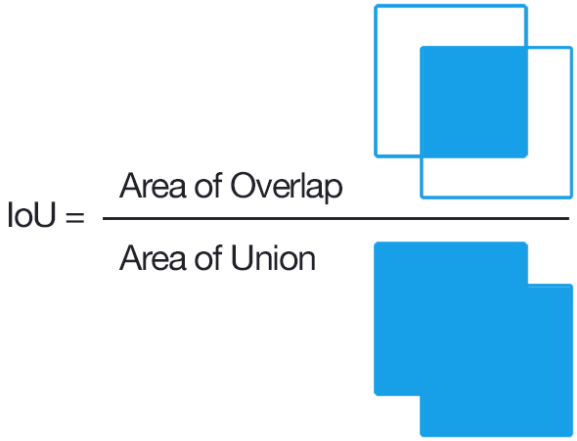


Figure 4.5: Intersection over Union visual representation, Area of overlap ($|A \cap B|$) divided by the area of union($|A \cup B|$) from: www.pyimagesearch.com

a detection on a target, this coefficient is commonly refereed to intersection over union due to it's mathematical formulation.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}. \tag{4.1}$$

In object detection the intersection over union can be easily understood through bounding boxes, with Figure 4.5 . In semantic segmentation would be described as:

$$MIoU = \frac{1}{|Classes|} \sum_{c \in Classes} J(GT_c, Output_c) \tag{4.2}$$

Where Classes are each of the classes represented in the segmentation, GT is the ground truth labels, Output is the output of the network and GT_c are the pixels of the Ground truth belonging to class c (same with output).

Mean Pixel Accuracy

An alternative metric to evaluate a semantic segmentation is to simply report the percent of pixels in the image which were correctly classified. The pixel accuracy is commonly reported for each class separately as well as globally across all classes. When considering the per-class pixel accuracy we're essentially evaluating a binary mask; a true positive represents a pixel that is correctly predicted to belong to the given class (according to the target mask) whereas a true negative represents a pixel that is correctly identified as not belonging to the given class.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

This metric can sometimes provide misleading results when the class representation is small within the image, as the measure will be biased. Mainly reporting how well you identify negative case (ie. where the class is not present).

4.2.2 Algorithms

In order to test the hypothesis we are going to evaluate the performance on the *deeplabv3* architecture.

Deeplabv3: *Deeplab's* third version, [27], improving from it's first version where the atrous convolution was introduced, by using the idea to create a pyramid with different dilation factor (See chapter 2).

4.3 Evaluation Protocol

Both experiments are extrapolated from [17], where they follow the same method in the object detection field.

4.3.1 Baseline of real datasets

For our first experiment we introduce the problem and see what are the baselines. Our goal is too see how much the domain shift affects the performance when a model is trained on only one dataset and tested on a different dataset.

This experiment aims at shedding a light on the initial discrepancies among the datasets and how much it affects the performance. This serves as a first step to see the improvements of future techniques and measuring the impact of each alternative.

4.3.2 How much real data do we need

This experiment attempts to analyze the performance obtained when there's little real data, we attempt to understand how much real data is actually needed when using synthetic data, understanding this could affect the decision making of future experiments, if we realize that only a small dataset is needed in order to achieve the best performance when used in conjunction with synthetic data, make many tasks feasible by only needing to manually label a small dataset. Rather than relying on thousands of real images with the related costs it implies.

In order to analyze this problem, we are going to train the model with two different approaches:

Mixture of real and synthetic data

This first approach serves as an initial exploration, what we are going to do is create different dataset corresponding of the whole synthetic dataset, being the MSS frames, and a proportion of the real dataset, 5%, 15% or 25%. With this new datasets we train a *deeplab* model and understand the results. See Figure 4.6.

Finetuning with a subset of the real data

With this experiment what we are going to understand is how are the learned weights on synthetic data generalizing to the real problem, in order to do so we are going to train the network with each of the synthetic datasets. With those learned models we finetune to the real data, but with only a proportion of the real data (5%, 15% or 25%). See Figure 4.7

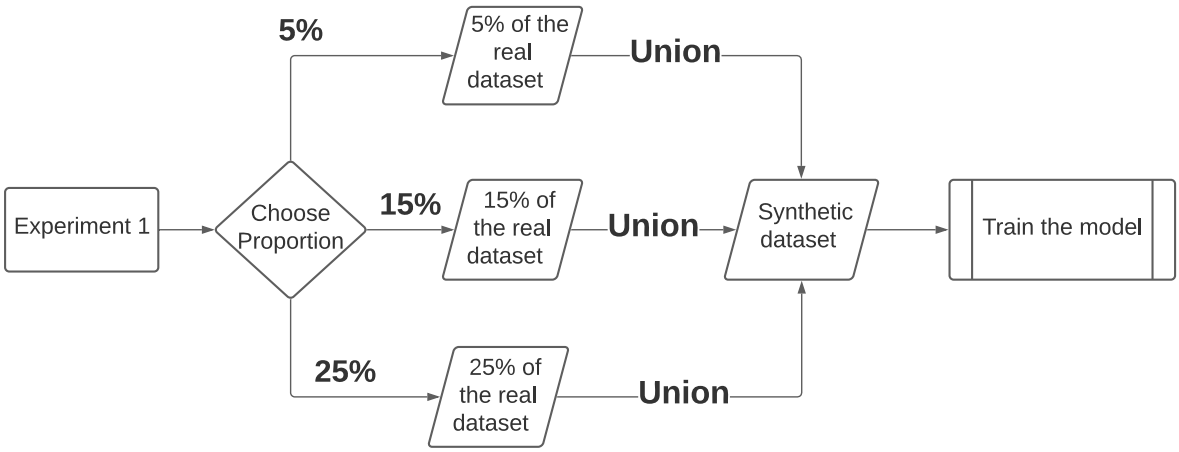


Figure 4.6: Mixture of real and synthetic data Block diagram

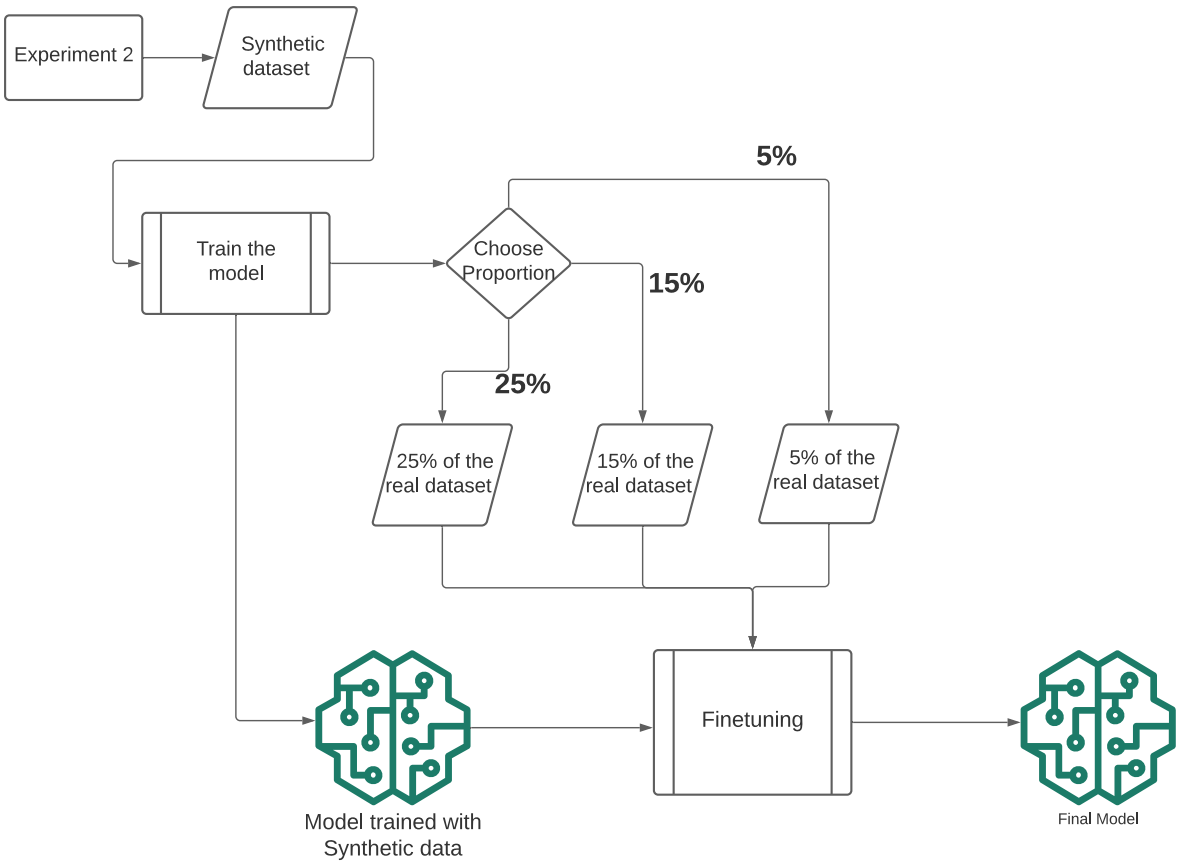


Figure 4.7: Finetuning with a subset of the real data Block diagram

Chapter 5

Experimental Results

In this chapter we will performe a comparative analysis between the proposed *MSS* dataset and the *Synthia dataset*. Our aim is to empirically measure the impact of each dataset. In order to do so we will first introduce the baseline of real and synthetic dataset. With an established baseline we will perform some experiments in order to measure the impact and to what extent.

We will conclude with a summary of conclusions gathered from the experiments in order to wrap up this Chapter.

5.1 Network configuration and experiments details.

In order to train the *DeeplabV3* network, we used an RSMprop optimizer, with an step scheduler, where every *step size* epoch, the learning rate would decay by a multiplicative factor γ. The parameters used are detailed in Table 5.1. The optimizer, learning rate, weight decay, scheduler, step size and γ are derived from similar projects [1] [2]

Optimizer	RSMprop
Learning rate	1e-4
Weight decay	1e-5
Scheduler	StepLR
Step size	25
γ	0.5
Batch size	8
Loss function	Cross entropy
Weighted loss	Effective number of samples [51]
Number of epoch	50

Table 5.1: Network configuration.

Finally as mentioned in section 4.3 we are going to be presenting portions of real data to train the network jointly or sequentially. As said proportions are dependent on the original size of the dataset. We include Table 5.2 to present the exact number of real images used for each proportion.

[1] https://github.com/pochih/FCN-pytorch
[2] https://github.com/kazuto1011/deeplab-pytorch

Dataset	Percentage	Size
Cityscapes	100	5000
Cityscapes	25	1250
Cityscapes	15	750
Cityscapes	5	250
Mapilliary	100	25000
Mapilliary	25	6250
Mapilliary	15	3750
Mapilliary	5	1250
MSS	1	89363
Synthia	1	220000

Table 5.2: Size of real datasets and proportions of the datasets

5.2 Baseline of real datasets

In this section our goal is to analyze the domain gap between different datasets, while using synthetic and real dataset there is a clear and distinctive domain gap, being the nature of the images.

However once we analyze the performance of training with different real datasets, we find that, while for a human eye they seem to be more related one another, the changes in the car models between one city and another, the architecture, the lighting conditions and the environment leads to a significant gap in performance.

5.2.1 Performance progress through the training curves

In this subsection we study the evolution of the training curves for each model's performance for each test set.

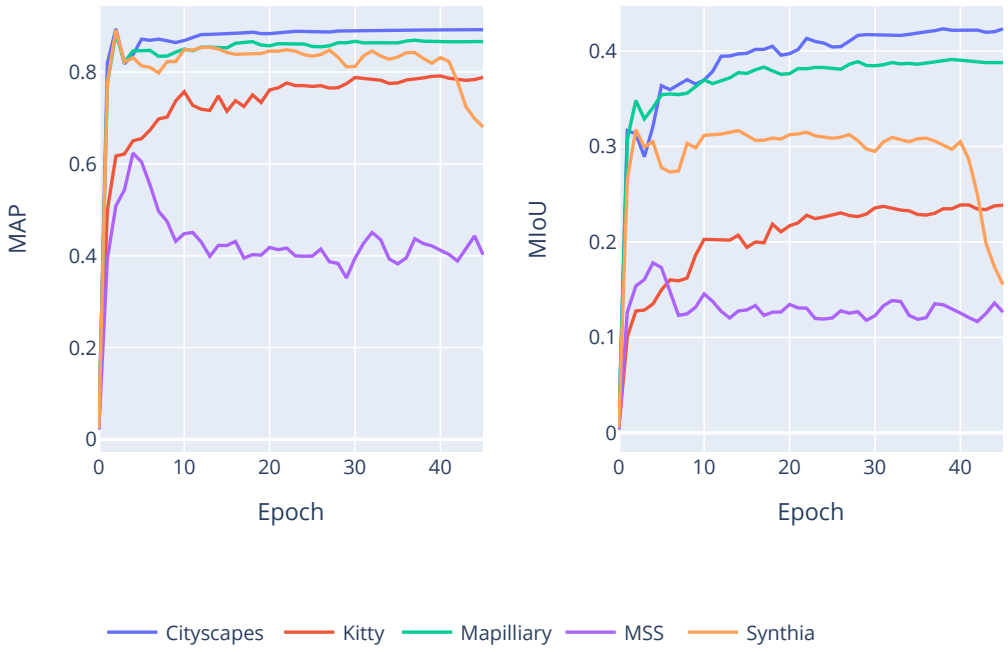
In Figure 5.1 we present the mean training evolution of 5 different random weight initialization for 45 epoch. This is: We have trained a *DeeplabV3* from scratch 5 times. With all the training curves we have computed the mean curve from those 5 runs.

We can see how synthetic datasets, *Synthia* and *MSS*, tend to achieve a better relative performance to the final performance on the early stages of the training. We believe this to be due to the amount of images present on the training sets.

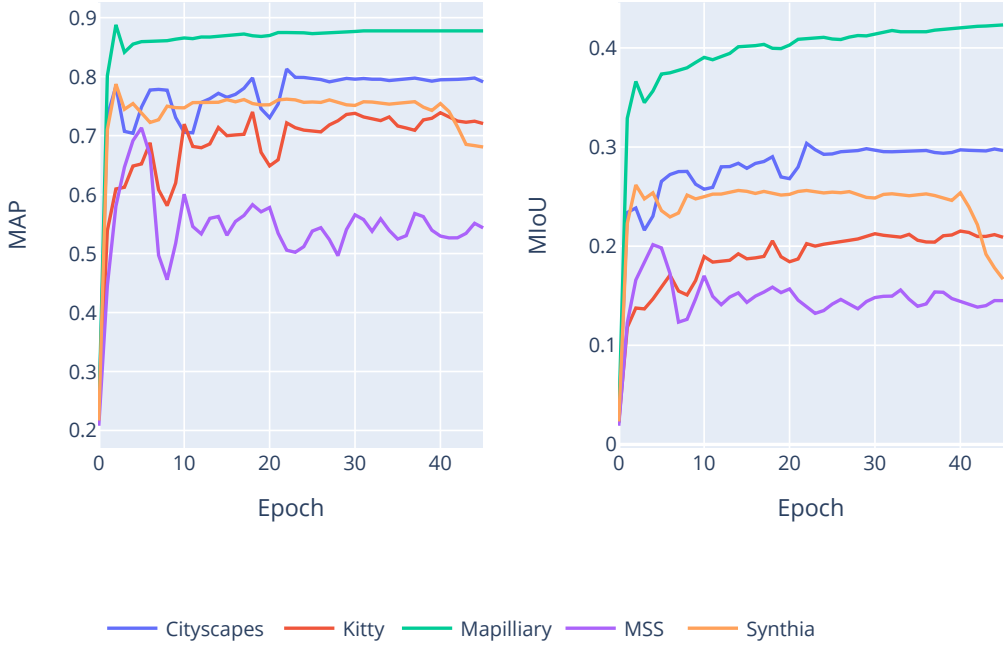
However, as the training progresses, the performance stalls or drops, we attribute this to the overspecialization of the network to the synthetic images, therefore loosing the generalization capabilities to the real domain.

We can see how the *MSS* suffers more from this overfitting than the *Synthia* dataset in early stages of the training. This can be seen in Figure 5.1, in the drastic drop after the first epochs. However, *Synthia* suffers from a steeper drop at the final epochs of the training.

Overfitting can be appreciated in the *Synthia* (orange) and the *MSS* (purple) curves, where once the top performance is achieved, the performance drops as the epochs progress. *MSS* presenting this problem earlier than *Synthia* may be due to the size of the dataset being over ten times smaller than the *Synthia's* or the lack of different environments in the *MSS* simulator, which contains one virtual city in day conditions, see Table 3.7.



(5.1.1) Training curves when testing on the *Cityscapes* test set



(5.1.2) Training curves when testing on the *Mapilliary* test set

Figure 5.1: In these Figures we can see how the training evolves for each of the training sets. 5 Trainings were averaged from random weights initializations

5.2.2 Final performance of each model

Now we will analyze the best achieved performance with each training set.

In Figure 5.2.2 we present the baseline of the real models, this is the best performance we could achive using the training set.

We would like to address how there is a clear drop in performance when testing on a different test set rather than it's own. The only one who consistently presents good results on all three test sets is Mapilliary.

We believe that the general good results of *Mapilliary* are mainly due to the big size gap between the datasets, refer to Table 3.7 for a size comparison.

In the next Figure 5.2, we include the results of training with synthetic datasets and testing in real test sets.

One thing to note is that here we present a clear domain shift, between the source data (Synthetic data obtained from simulators) and the target data (Real data obtained with cameras). Although initially we may think that due to the domain shift, these new results should be considerably worse. However, we can see that there is not such a clear difference. Obviously, the best performance is obtained through training and testing with the respective sets of the same dataset.

As authors in [17] already addressed in object detection, there is a persistent domain shift even between real datasets. Due to the car models, lighting conditions and overall structure of the captured environment, there are clear discrepancies between real datasets, see Figure 2.9 for some visual examples of the datasets, where clear differences can be appreciated.

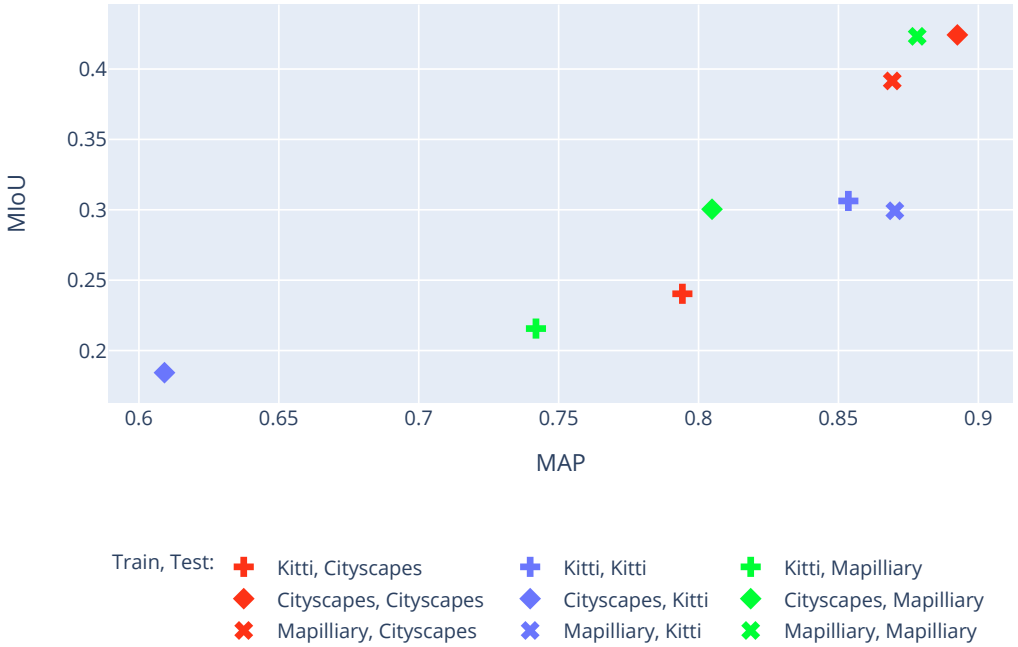
The domain shift is so drastic that when enough data is provided, such as with the *MSS* dataset, the performance of the model trained with synthetic data competes favorably with a model trained with real data, *MSS* performance on the *Mapilliary* is almost 5% better than the performance of Kitty on the same set).

Training on small real datasets, like Kitty, yields models less generals than the one trained with synthetic data. While this is something already known in the literature, ([17] already dove into this problem using synthetic data on object detection). To the best of our knowledge there's no previous study of this problem in semantic segmentation and how much it differs from real to synthetic.

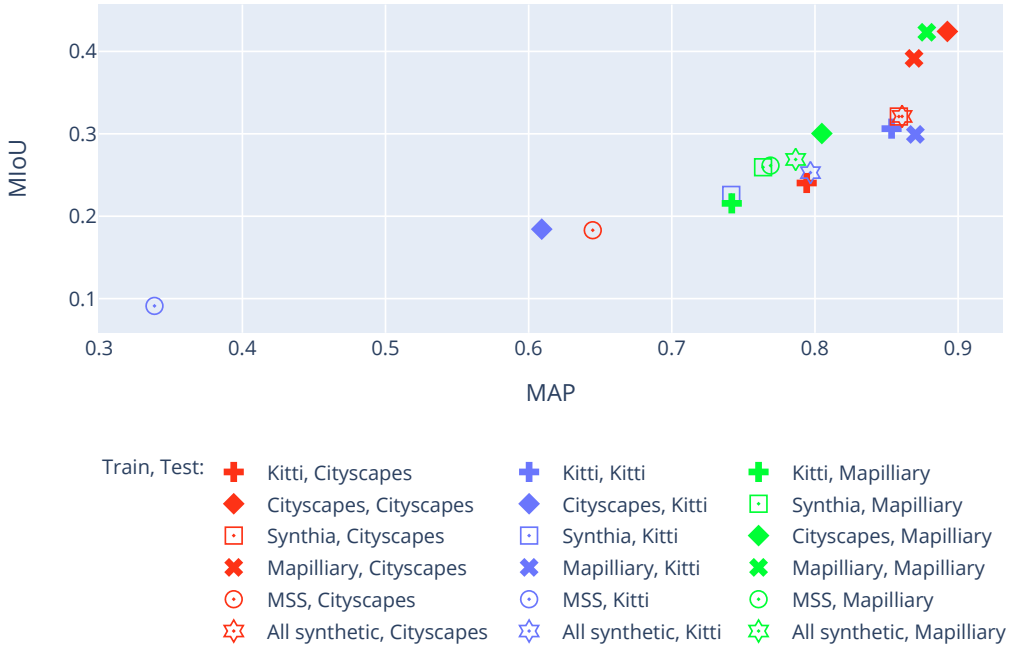
In Figure 5.2 we decided to include a new training set called *All synthetic*, this train set consist on using the full *MSS* and *Synthia* datasets jointly to train. This training set will appear in all future experiments.

Our goal with this new training set is to prove that the power of synthetic data relies on it's size, rather than it's specificity.

As a first result for this goal, in Figure 5.2 this new training set, *All synthetic* (star), induces a boost in the *Mapilliary* dataset without harming the performance on the *Cityscapes* dataset, compared with training with the *Synthia* dataset. This improvement is persitent when compared with the *MSS* dataset.



(5.2.1) Baseline performance of the real datasets.



(5.2.2) Baseline performance of all datasets.

Figure 5.2: Baseline performance. Each marker shape indicates the training set used, filled markers means it's a real dataset, empty markers means it's a synthetic dataset. Color indicates the test set used for the metrics obtained. All synthetic is a new training set using all synthetic images from *Synthia* and *MSS* for training.

From this experiments three main results can be extrapolated.

First, the important thing is the size of the dataset, when using a synthetic dataset we need many more synthetic images to achieve the same performance. See Table 5.2

Second, as there is a domain gap among all datasets, applying some sort of transfer learning like finetuning or adding some data from the target domain is essential despite the origin of the original training data. As can be seen in Figure 5.2.2, where we can see how real datasets lack abstraction to perform well in other test sets.

Third, combining synthetic datasets seem to bring a better performance than using one source. See Figure 5.2

5.3 How much real data do we need

As introduced in the previous chapters, when enough data on the training set is available, there's not such a need for real data to be used in order to obtain good results. However, when there's few to no data synthetic data can drastically improve the performance, as it's presented in this and the following sections.

In this experiment we aim at mimicking scenarios where there's little available data from the source to train. In order to measure the impact of each synthetic dataset in this scenarios we will train with a percentage of the real data varying from 5% up to 25% of the original dataset mixed with the full synthetic dataset. In Figures 5.3.1 and 5.3.2 we compare the results obtained when mixing different proportions of real and synthetic data.

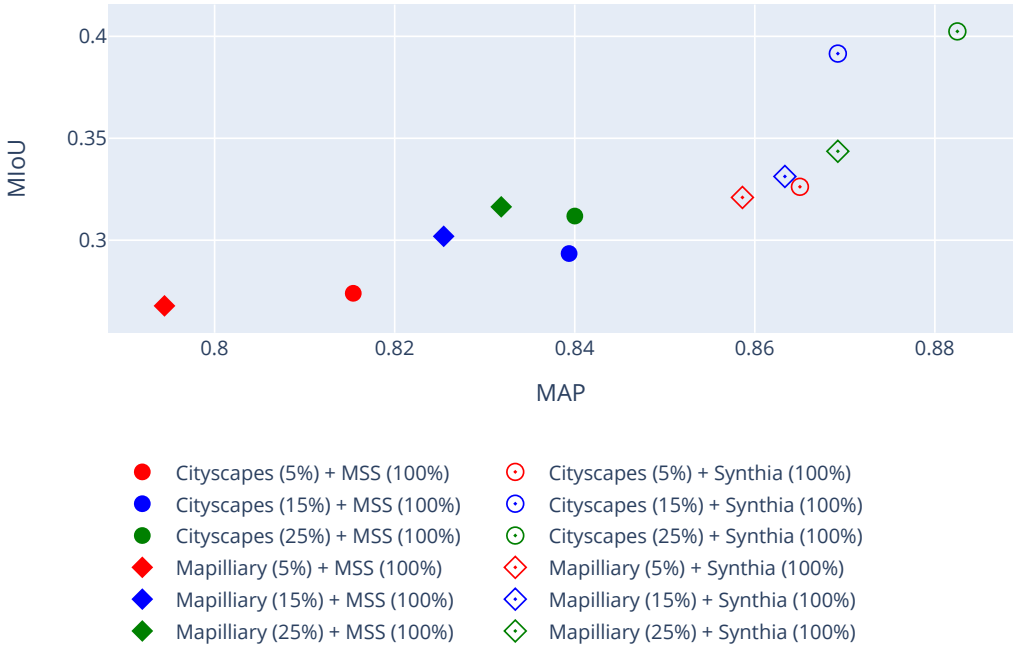
When comparing *MSS* the proposed dataset with *Synthia*, in Figure 5.3.2 can be seen how on the *Mapilliary* set *MSS* provides slightly better models. This is noticeable because all *MSS* markers (filled) appear to the top right of the *Synthia* markers (empty) for each mixture.

However, when testing on the *Cityscapes* dataset, we find the opposite, where all *Mapilliary* present better results than the *MSS*

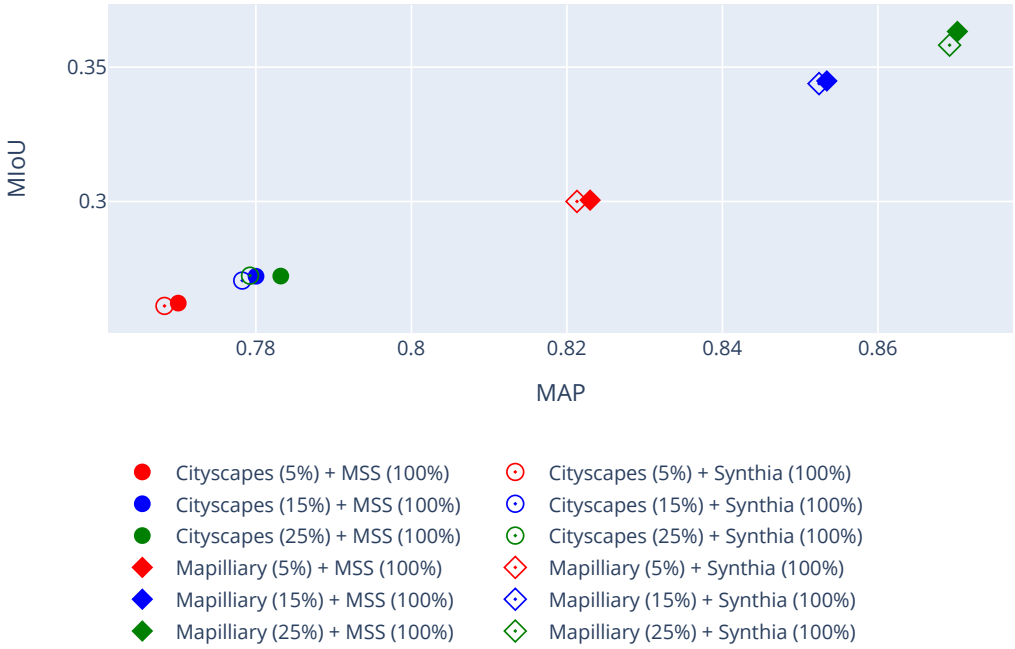
We believe this to be due to the closer proximity *MSS* and *Mapilliary* share compared to *Synthia* in the percentage of pixels per class represented, see Table 3.7.

Following this experiment we will proceed by comparing these mixtures of real and synthetic data to the baselines of the real datasets. In Figure 5.4 we present the results of the models trained with synthetic data and only a fraction of the real data against the final performance with the whole dataset.

In Figure 5.4 we can see how with only a small fraction of the real dataset the results are almost as accurate as using the whole real dataset. This implies that a rather huge amount of cost saving could be achieved in data annotation.



(5.3.1) Results of the models tested on the *Cityscapes* test set



(5.3.2) Results of the models tested on the *Mapilliary* test set

Figure 5.3: Performance when synthetic data is mixed with real data. Color indicates proportion of real data (red 5%, blue 15%, green 25%), shape indicates real data used (circle *Cityscapes*, diamond Mapilliary), fill of the marker indicates synthetic data used (filled MSS, open Synthia)

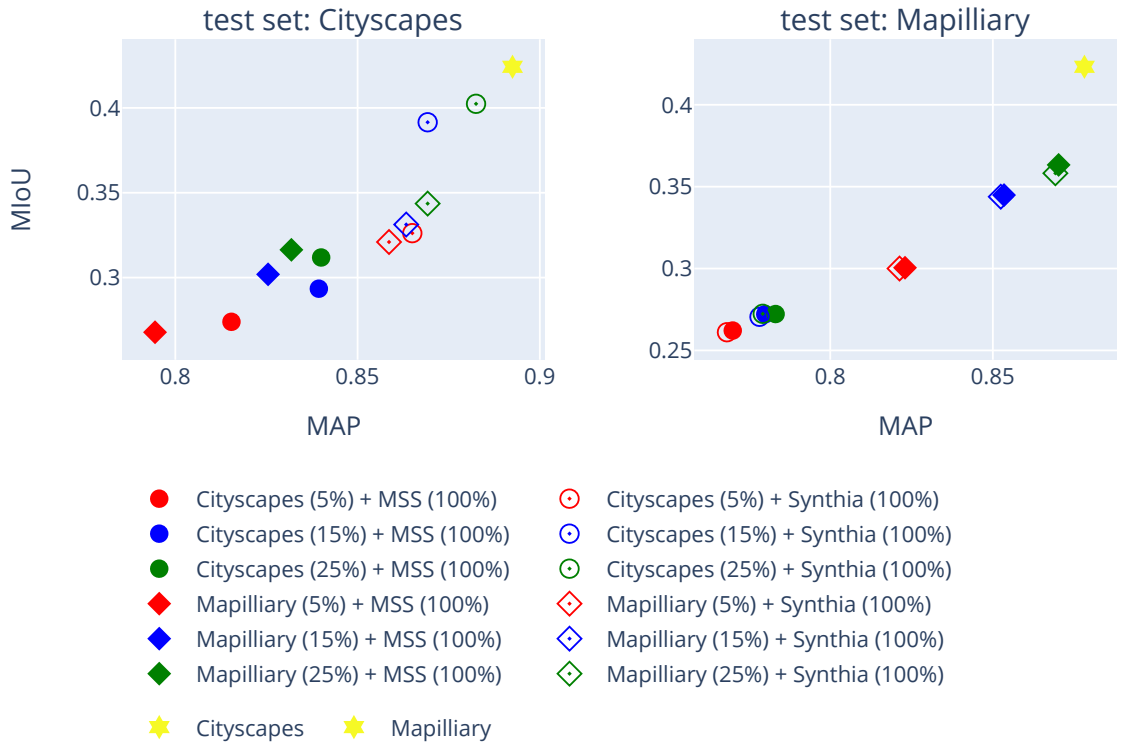


Figure 5.4: Comparison of training with hybrid synthetic and real train sets with original train set. Tested on real test sets.

Four things can be concluded from this results:

One of the first things we can see is how the more real data is included the better the results obtained. The performance is directly correlated with the percentage of the real dataset included. See Figure 5.3 This serves as a good indicator of initial expectations, the more data included the better the results are.

When including real data i.i.d. from the test set, induces a bigger and broader boost in performance than including real data from different domains, such as another real dataset. Although including other datasets is be beneficial if enough data is present, greater boosts are achieved when including data from the same source. See Figures 5.3.1, 5.3.2

Initial performance of the synthetic model on it's own is a good indicator of how good it'd extrapolate when mixed with real data. We can see how the *MSS* competes favorably when tested in the *Mapilliary* dataset against models trained with Synthia, see Figure 5.3.2. On the other hand, *Synthia* has the edge when tested on *Cityscapes*, see Figure 5.3.1.

When including real data from an external source for example *Cityscapes* data for testing on the *Mapilliary* dataset, we can see how the inclusion of hundreds of images have a marginal impact on the final performance of the model, see Figure 5.3.2 and

Table 5.2 for the sizes of each step. The true potential is when thousands of images are included. See Figure 5.3.1, the inclusion of a proportion of Mapilliary induces a greater boost in performance due to it's size.

5.4 Finetuning with real data

Similar to the previous experiment, we aim at simulating an scenario where there's little available data from the source to fully train a network. In contrast with the previous scenario, where full access was granted to the synthetic data and enough resources are available to train, finetuning a pretrained model needs less resources. When finetuning only the real set is used, thus, reducing considerably the amount of computations.

Furthermore, when used hybrid training sets, we expect the model to learn the general concepts from simulated images, and use the real samples to adapt. However, there is no scheduling nor structure in the hybrid training approach, samples from synthetic and real data are presented at a random pace. Therefore, there is no guaranty that such expectations are met.

In order to perform a more structured experiment, we take a transfer learning approach. The model is first trained with a synthetic set, and then fine-tuned using each of the real training sets. We use the same ratios defined in the previous section. Similarly, the tests take place on the real data only.

In Figures 5.55.6 we present the results of this experiment.

Finetuning pretrained models with only a portion of the real dataset, *Cityscapes* in 5.5 and *Mapilliary* in 5.6, yields models which compete favorably with only 25% of the real dataset. The baseline is presented as yellow sandclocks.

Despite *Mapilliary* having the best baseline performance, see Figure **??**. We can see how these finetuned models provide a significant improvement in the baseline. There's a boost in performance both in generalization and specificity.

Generalization can be measured by the performance on the *Cityscapes* and *Kitti* test set is improved to the baseline.

Specificity can be measured by the performance on the *Mapilliary* test set.

It's clear how in both aspect the finetuned models provide better results than the baseline. This not only proves how useful synthetic data is, but completely changes the paradigm when tackling semantic segmentation.

Also we would like to remark how the new dataset, *All synthetic* yields models which are consistently better than the ones from only one synthetic source.

Settled the generalities, now we will focus on the individual results obtained when finetuning with the *Cityscapes* dataset.

Something else to remark from 5.5, is how mixed training gives us a brief vague intuition of how the model will perform once we apply transfer learning, we can see how in this example the pretrained model in the *Synthia* datasets yields better model than the one from *MSS*, see Figure 5.4. However, when looking at Figure 5.6, we can see how *MSS* consistently brings a better generalization to the *Mapilliary* test set when real images are included. Therefore, we can make an argument that whatever the initial performance on the real test set models trained with synthetic data, see Figure 5.2,

will be improved when applying transfer learning but there's a consistent gap between models which was originated since the first experiment of the chapter.

Regarding Figure 5.6, we can see how when finetuning from *Mapilliary* a broader gap is present among proportions, this can be attributed to two factors, every step in the introduction of real data (5%,15%, 25%) introduces approximately 2500 real images, this in contrast with 5.5, which each step would only include around 500 real images, creates a bigger impact due to only the amount of data introduced, which is the main premise of this work, the more amount of data the better. We can see, how in this case this is clearly exemplified through a broader color dispersion in the graph (representing the different performance on different train sizes).

We can see how with the inclusion of only a 15% of the *Mapilliary* dataset we obtain models which can compete favorably with the one trained with the full set, this is even more surprising noting how the *Mapilliary* dataset was the one with the best initial performance on our first experiment, see Figure 5.2.2.

Finally in Appendix C we touch on catastrophic forgetting [52], diving on how when finetuning the model trained on the *Kitty* dataset (real dataset) yields better results in the *Cityscapes* and *Mapilliary*. Never the less, harms the performance on the *Kitty* test set. Some authors purpose many models to alleviate this problem, although it's out of the scope of this project.

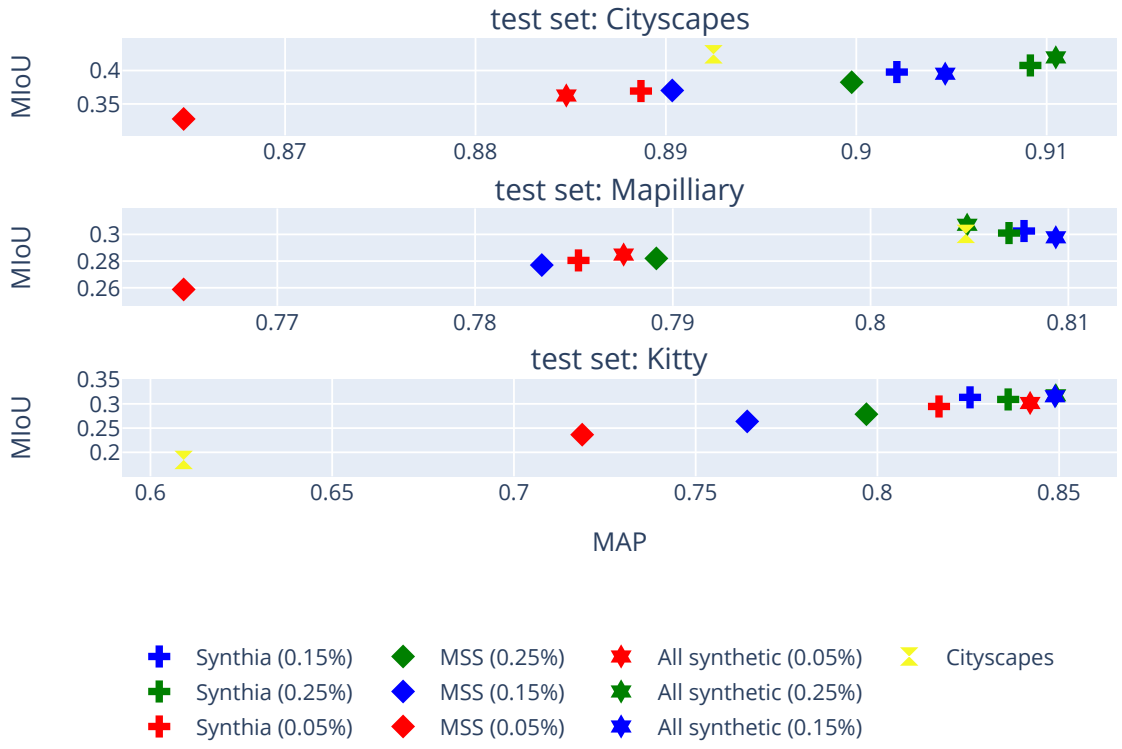


Figure 5.5: Performance of the models finetuned with portions of *Cityscapes* tested on the *Cityscapes*, *Mapilliary* and Kitty test sets. *All synthetic* is a pretrained model from the *MSS* and the *Synthia* dataset.

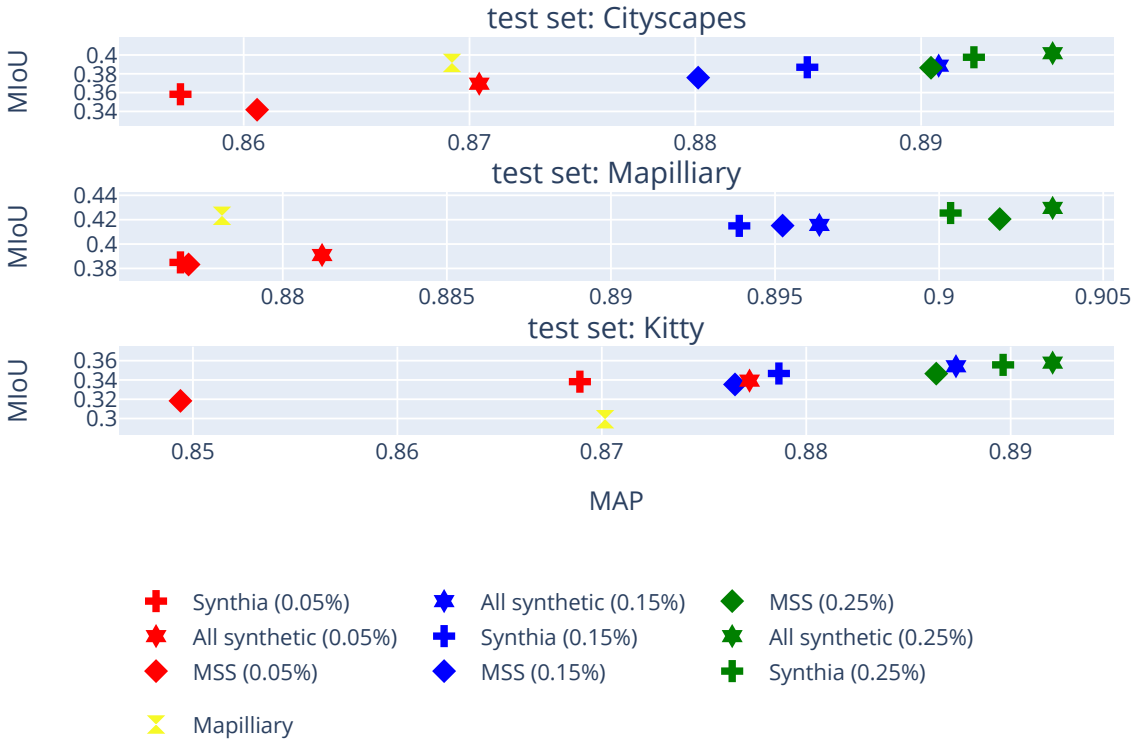


Figure 5.6: Performance of the models finetuned with portions of *Mapilliary* and *Cityscapes* tested on the *Cityscapes*, *Mapilliary* and Kitty test sets. *All synthetic* is a pretrained model from the *MSS* and the *Synthia* dataset.

5.5 Summary

In order to sum things up we would like to present a brief list of ideas extrapolated from these results:

The more the merrier, when training with synthetic data, we see that the size of the dataset and the variability presented in the dataset is tightly related to it's performance, whether it is through itself or by mixed training with real data or by a final step of transfer learning to real data. Therefore using different synthetic datasets to the task can be a good approach, see Figures 5.4, 5.55.6.

There is a persistent domain gap between all datasets. Not only between real and synthetic sets. This was already introduced by [17] in the context of object detection. In Figure 5.2.2 we include the performance of each training set, we can see how in some test sets synthetic data from scratch outperforms real data.

It is better to finetune than to train with a mixture of real and synthetic data. We believe this to be caused because when using hybrid training sets there is no structure

of the training. This is, all samples come at random pace. We would like the network to abstract generalities from synthetic data and then refine it's weights with real data to be able to extrapolate to real scenarios. However, when using hybrid training sets may not be the case. Due to having more synthetic images than real ones, the network could extrapolate that the core future problem is to be really specific in the synthetic samples, hence, use real images to be more accurate on the synthetic domain.

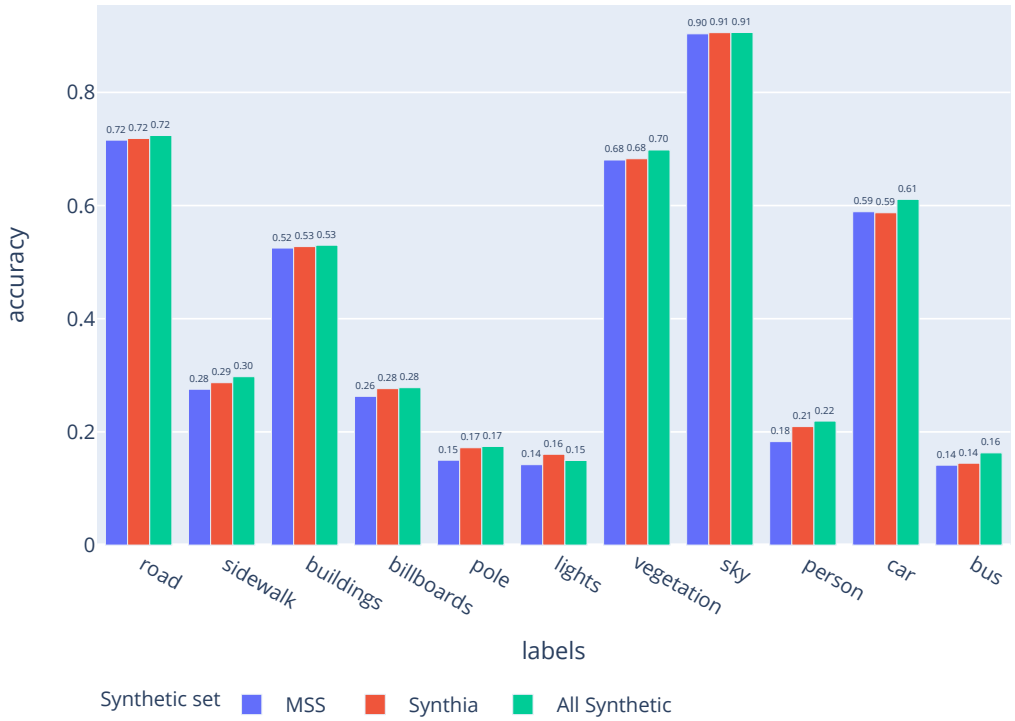
In contrast, when using transfer learning techniques such as finetuning the network only optimizes to the real domain. Through the loss in real images we are reinforcing the network to abstract the generalities it has obtained in the synthetic domain and refine them to the real domain. See Figures 5.4, 5.55.6.

When transfer learning with a model trained in real images catastrophic forgetting may occur, see Appendix C. As we've seen when finetuning from the *Kitti* model, the final performance ends up being worse in the original domain than before due to specification on the target domain. This is, the finetuned model performs really well in the *Cityscapes* and *Mapilliary* test set but drops drastically it's performance on the *Kitti* test set. This makes synthetic data much more reliable than small real sets. Due to the size of the initial training set synthetic datasets are more general.

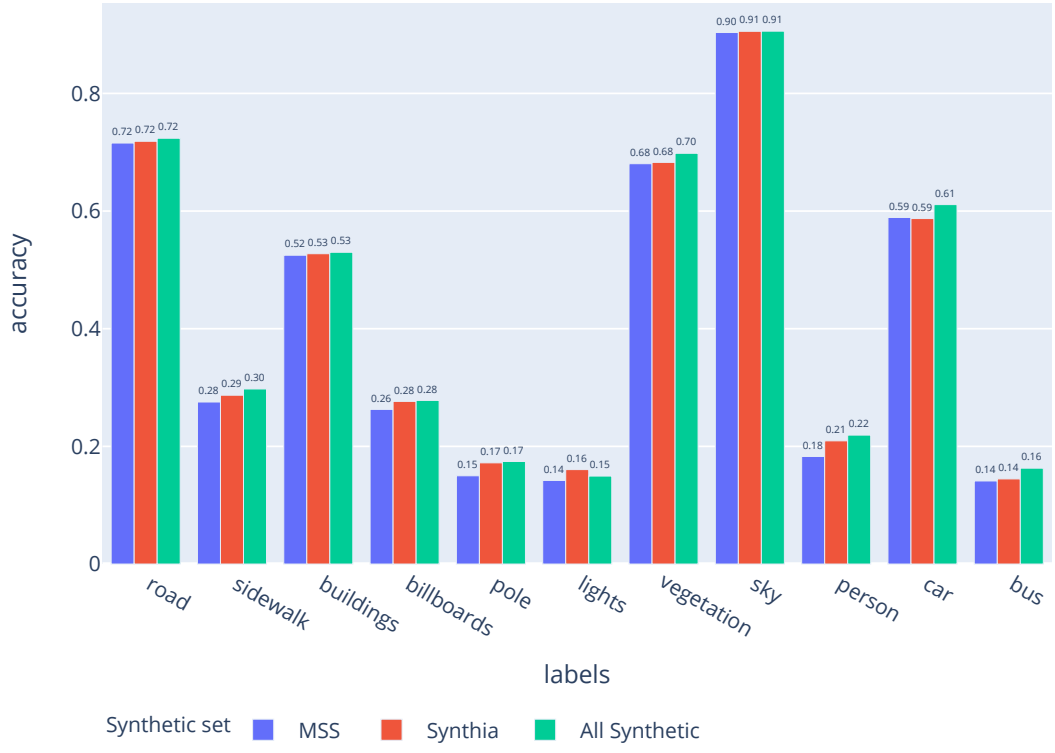
When introducing real data, the difference in range of the hundreds or below of samples produce little effect on the final performance of the model, see Table 5.2. As we've seen when finetuning with the *Cityscapes* dataset the small additions of a few hundred of samples to the training set produced jumps in performance of hundredths, see Figure 5.3.2. However, if we manage to include thousands of images, this effect is multiplied, as we saw when finetuning with the *Mapilliary* dataset. See Figures 5.55.6.

Synthetic dataset similarity with the real set is an important metric for the final performance if no more samples can be introduced. The similarity may be measured by the initial performance when trained from scratch on the target set. This performance gap is carried throughout all our experiments. See Figures 5.2.2, 5.4, 5.55.6. *MSS* is closely related to *Mapilliary* and *Synthia* to *Cityscapes*. See Table 3.7. Even though the gap in performance is small the models trained with *MSS* produce better results on the *Mapilliary* test set than it's counterpart, and the other way around with the *Synthia* test set. However, including both sets, *All synthetic*, produce better results than each of the individuals in all test sets. See Figures 5.2.2, 5.4, 5.55.6.

In order to illustrate the advantages of using *All synthetic* images we include the performance per class of the best models obtained in the experiments in Figure 5.7. We can see how *MSS* brings a better performance on cars, vegetation and sky compared with *Synthia*. *Synthia* has a better performance on smaller objects such as lights, poles and pedestrians. Using both synthetic datasets, *All synthetic*, performs better than both in every class. Specially we see a bigger boost in cars, pedestrians, sidewalk and buses.



(5.7.1) Models were obtained from finetuning pretrained models on synthetic sets with 25% of the *Mapilliary* train set. Tested on *Mapilliary* test set.



(5.7.2) Models were obtained from finetuning pretrained models on synthetic sets with 25% of the *Cityscapes* train set. Tested on *Cityscapes* test set.

Figure 5.7: Per class performance of the best models using synthetic datasets. Models were obtained from finetuning pretrained models on synthetic sets with 25% of real data. Tested on the corresponding real test set.

Finally in 5.8 and 5.9 we include the comparison between the label maps obtained from the network finetuned from pretrained weights from *All synthetic* with *Cityscapes* and *Mapilliary* respectively. We can see how although on poles and lights the performance is not the best, regarding general structure of the image, cars, vegetation, buildings, sky and road, the network performs with a significant precision.

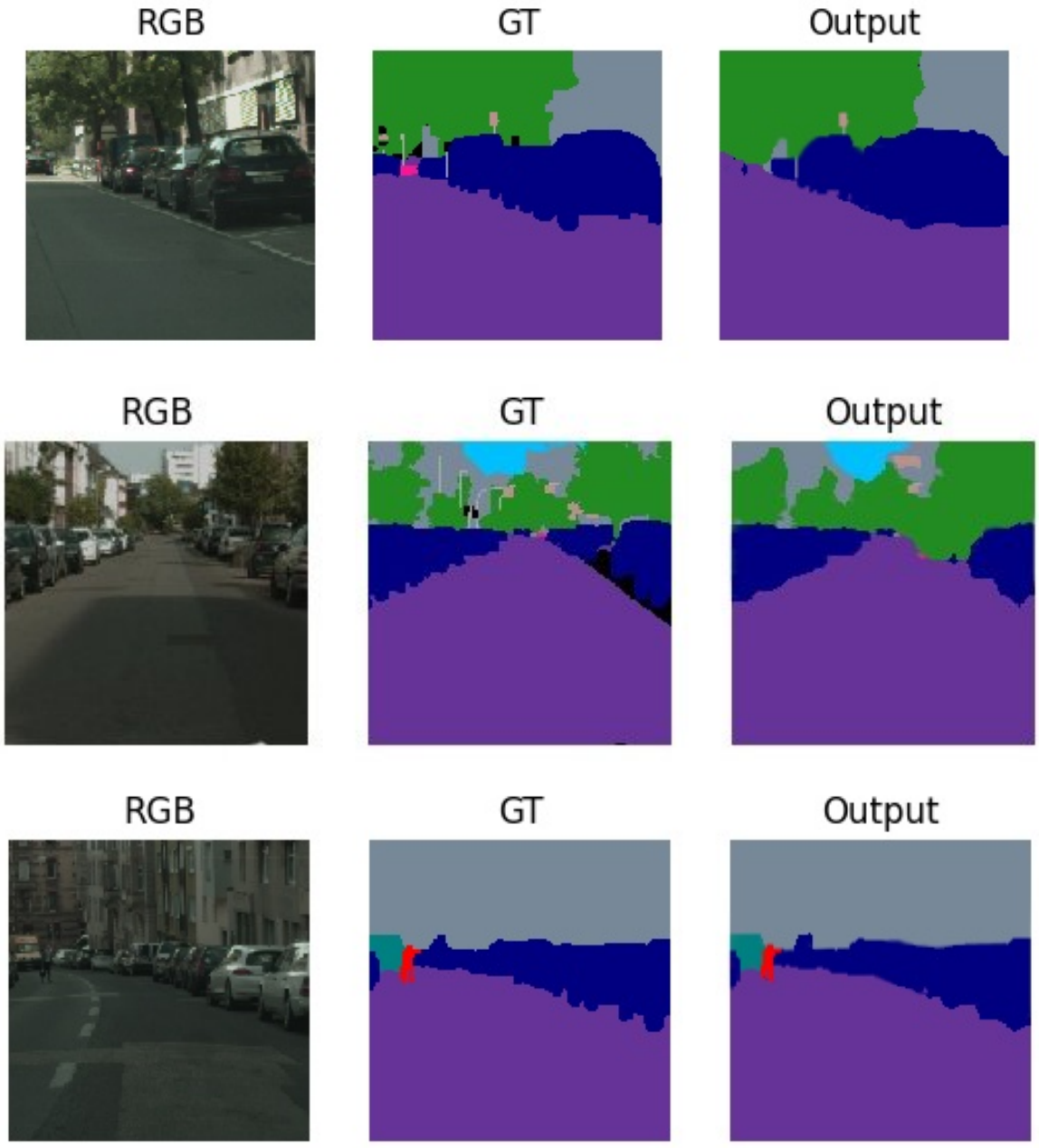


Figure 5.8: Sample results from the *Cityscapes* dataset from the model trained with *All synthetic* and finetuned with a 25% of the *Cityscapes* dataset

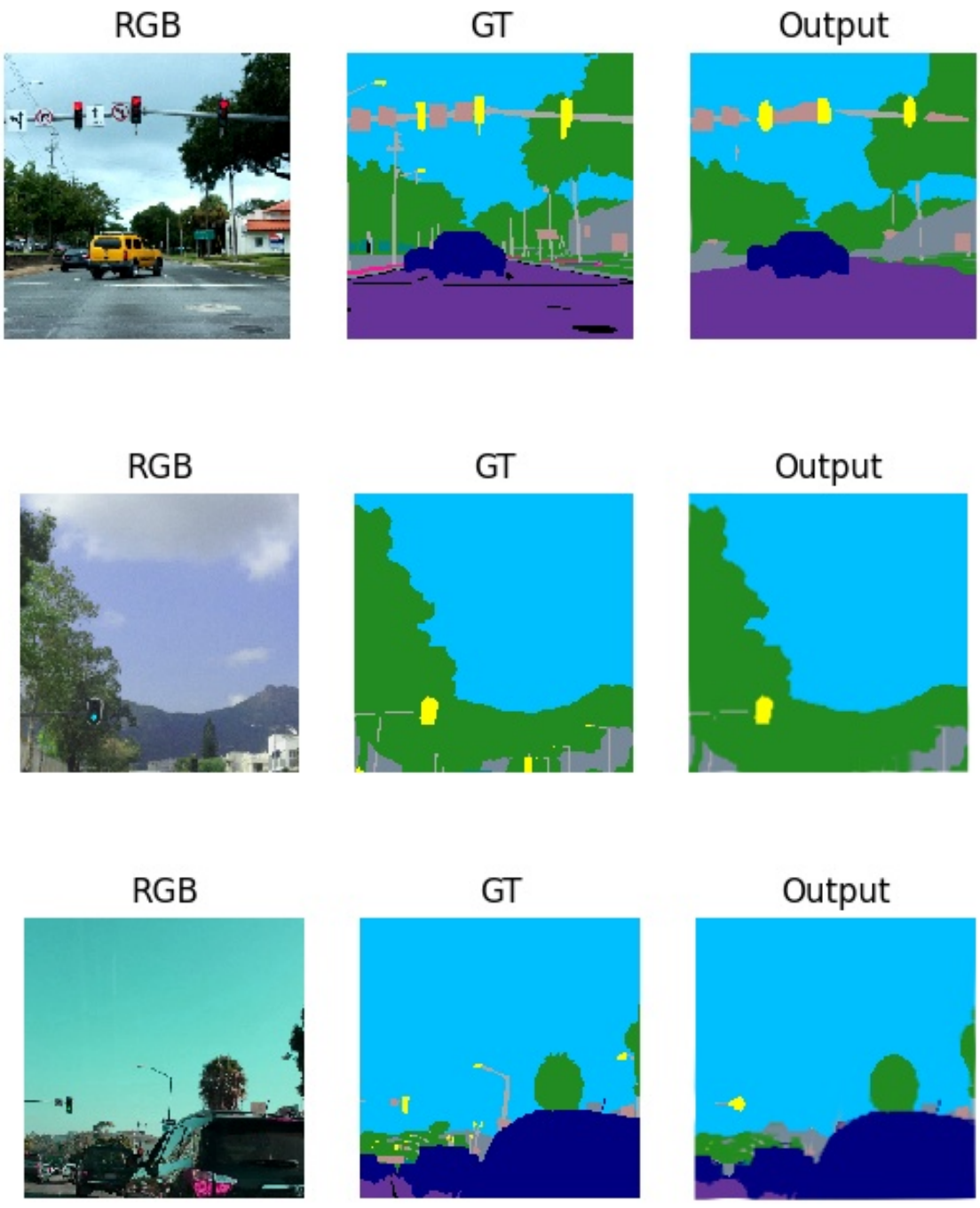


Figure 5.9: Sample results from the *Mapilliary* dataset from the model trained with *All synthetic* and finetuned with a 25% of the *Mapilliary* dataset

Chapter 6

Conclusions and future work

6.1 Conclusions

Going through the work we find that synthetic data is useful by its own and in combination with real data. We see that where synthetic data shines most is in scenarios where there's little real data to fully train a model. From this work we can extrapolate the following conclusions:

When generating synthetic datasets if the goal task is known generating scenes with similar biases and compositions, such as point of view of the source images, will help the generalization to the real domain.

Scenes obtained through static cameras on posts and wearable cameras of cars yield better results compared with others with different structure such as cameras on helicopters.

Smaller objects tend to be punished harder. Smaller objects such as pedestrians have an inherent bigger domain gap and are not stable through time or location, leading to a harder extrapolation. In contrast, broader structures such as cars, roads and buildings, although different, share biases and general structure with real datasets.

There is a persistent domain gap even among real datasets. In some instances broader than the one found between synthetic and real sets. This makes synthetic data interesting throughout any condition due to finetuning being one of the most popular techniques in the field. Introducing finetuning from pretrained sets on synthetic data is a good approach and should be heavily considered one there's not a huge staple set available.

Although synthetic data is useful, the amounts of synthetic images needed to extrapolate to real scenarios needs to be as big as possible. Small sets ($500 \leq$) of synthetic data leads to almost random classifiers. We find that we need a size from tens of thousands upward. In addition, variability makes the network focus less on biases of each of the datasets, categories and images. Therefore, needs to find common patterns of shape and location, rather than focusing on color and texture.

Training with all synthetic datasets at the same time yields better models than the ones trained with synthetic data from only one source.

Using models pretrained on real data leads to a drop in performance on the original dataset due to catastrophic forgetting. When using synthetic data models tend to

generalize better than the ones with real data.

When using synthetic data, transfer learning approaches work better than using hybrid training sets with synthetic and real data.

6.2 Future work

Based on the results obtained we purpose to follow the work by:

- Implement the correction script of the GT from the MSS tool in SQL in order to speed up the process and benefit from it's inherent parallelization protocols from Postgress [53].
- Expand the MSS dataset with new virtual cities from the toolkit which are currently on development.
- Expand the MSS dataset by including a subdataset of domain randomization. [9]
- Ablation study on the impact of different aspects of the network, as the learning rate, optimizer, loss function and normalization to the generalization of synthetic datasets.
- Include curriculum learning strategies through the synthetic data generated from the MSS tool and measure if they are reliable and useful [[19], [21]].
- Advance on the domain adaptation approaches by implementing and training a style transfer network to generate hyper realistic images [[14], [15]].
- Implementing a new architecture for semantic segmentation specialized in binding the gap between real and synthetic images. Possible approach may involve including into the loss an adversarial component to force the network not to rely on synthetic or real latent information and use only generalities among both domains [20].
- Expand on the usage of synthetic data to improve the performance in scenarios where full datasets are available, closely related to previous two items.

Bibliography

[1] Z. R. Struzik and A. Siebes, "The haar wavelet transform in the time series similarity paradigm," in *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '99, (Berlin, Heidelberg), p. 12–22, Springer-Verlag, 1999. 1

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004. 1

[3] S. Brahnam, L. C. Jain, L. Nanni, and A. Lumini, *Local Binary Patterns: New Variants and Applications*. Springer Publishing Company, Incorporated, 2013. 1

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, 2005. 1

[5] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995. 1

[6] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002. 1

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, Ieee, 2009. 1

[8] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 3234–3243, 2016. 1, 14

[9] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2018-June, pp. 1082–1090, 2018. 1, 2, 12, 40, 68, 82, 85

[10] V. Pazmino, "Análisis automático de vídeo simulado con sistemas multicámaras basados en unity," *Trabajo de Fin de Grado, UAM*, 2020. 2, 14, 39

[11] F. Terry, "Análisis automático de vídeo simulado con sistemas multicámaras basados en unity," *Tabajo de Fin de Grado, UAM*, 2020. 2

[12] M. González, "Sistema multi-cámara distribuido basado en unity," *Tabajo de Fin de Grado, UAM*, 2020. 2, 15

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. 2, 8, 10

[14] Y. Chen, W. Li, X. Chen, and L. V. Gool, "Learning semantic segmentation from synthetic data: A geometrically guided input-output adaptation approach," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 1841–1850, 2019. 2, 10, 68

[15] G. Architecture and D. Architecture, "Learning from synthetic data : Addressing domain shift for semantic segmentation," pp. 3–5, 2018. 2, 11, 12, 68

[16] L. Gatys, A. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv*, 08 2015. 2, 8

[17] F. E. Nowruzi, P. Kapoor, D. Kolhatkar, F. A. Hassanat, R. Laganiere, and J. Rebut, "How much real data do we actually need: Analyzing object detection performance using synthetic and real data," *arXiv*, 2019. 2, 5, 6, 22, 25, 29, 40, 49, 54, 61, 82, 83

[18] G. Georgakis, A. Mousavian, A. C. Berg, and J. Košecká, "Synthesizing training data for object detection in indoor scenes," *arXiv*, 2017. 2, 5, 40, 82

[19] S. Hinterstoisser, O. Pauly, H. Heibel, M. Marek, and M. Bokeloh, "An annotation saved is an annotation earned: Using fully synthetic training for object instance detection," *arXiv*, 2019. 2, 5, 6, 40, 68, 82, 84, 86

[20] M. Biasetton, U. Michieli, G. Agresti, and P. Zanuttigh, "Unsupervised domain adaptation for semantic segmentation of urban scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 2, 10, 68

[21] Y. Zhang, P. David, H. Foroosh, and B. Gong, "A curriculum domain adaptation approach to the semantic segmentation of urban scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 1823–1841, 2020. 2, 9, 11, 12, 68

[22] Y. Chen, W. Li, and L. V. Gool, "Road: Reality oriented adaptation for semantic segmentation of urban scenes," *arXiv*, 2017. 2, 9, 11, 12

[23] X. Guo, W. Chen, and J. Yin, "A simple approach for unsupervised domain adaptation," 12 2016. 5

[24] J. Jeong, T. S. Yoon, and J. B. Park, "Towards a meaningful 3d map using a 3d lidar and a camera," *Sensors*, vol. 18, no. 8, 2018. 6

[25] K. Sun, Y. Zhao, B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, and J. Wang, "High-resolution representations for labeling pixels and regions," *CoRR*, vol. abs/1904.04514, 2019. 6

[26] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015. 7

[27] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *European Conference on Computer Vision, (ECCV)*, 2018. 7, 8, 49

[28] J. Zhang, C. Lu, J. Wang, L. Wang, and X.-G. Yue, "Concrete cracks detection based on fcn with dilated convolution," *Applied Sciences*, vol. 9, no. 13, 2019. 7

[29] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1422–1430, 2015. 8

[30] H. Dong and Y. Yang, "Towards a deeper understanding of adversarial losses," *CoRR*, vol. abs/1901.08753, 2019. 8

[31] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1365–1374, 2019. 9

[32] J. Cho and M. Lee, "Building a compact convolutional neural network for embedded intelligent sensor systems using group sparsity and knowledge distillation," *Sensors*, vol. 19, no. 19, 2019. 9

[33] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker, "Learning to adapt structured output space for semantic segmentation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7472–7481, 2018. 10, 12

[34] G. Neuhold, T. Ollmann, S. Rota Bulò, and P. Kontschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *International Conference on Computer Vision (ICCV)*, 2017. 13

[35] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 13

[36] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 13, 14, 82

[37] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in Neural Information Processing Systems*, vol. 1, Morgan-Kaufmann, 1989. 13

[38] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European Conference on Computer Vision (ECCV)*, pp. 102–118, Springer International Publishing, 2016. 14

[39] D. Lee, K. Chen, K. Liou, C. Liu, and J. Liu, "Deep learning and control algorithms of direct perception for autonomous driving," *CoRR*, vol. abs/1910.12031, 2019. 14

[40] S. Karanam, M. Gou, Z. Wu, A. Rates-Borras, O. I. Camps, and R. J. Radke, "A comprehensive evaluation and benchmark for person re-identification: Features, metrics, and datasets," *CoRR*, vol. abs/1605.09653, 2016. 14

[41] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017. 14, 82

[42] W. Li, C. Pan, R. Zhang, J. Ren, Y. Ma, J. Fang, F. Yan, Q. Geng, X. Huang, H. Gong, W. Xu, G. Wang, D. Manocha, and R. Yang, "AADS: augmented autonomous driving simulation using data-driven algorithms," *CoRR*, vol. abs/1901.07849, 2019. 14

[43] B. Hurl, K. Czarnecki, and S. L. Waslander, "Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception," *CoRR*, vol. abs/1905.00160, 2019. 14

[44] A. Tao, K. Sapra, and B. Catanzaro, "Hierarchical multi-scale attention for semantic segmentation," *CoRR*, vol. abs/2005.10821, 2020. 16

[45] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020. 29

[46] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets computer vision: Efficient data generation for urban driving scenes," *International Journal of Computer Vision*, vol. 126, pp. 961–972, 2018. 40

[47] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995. 43

[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019. 43

[49] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018. 45

[50] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, Austin, TX, 2010. 46

[51] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 51

[52] M. McCloskey and N. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology of Learning and Motivation - Advances in Research and Theory*, vol. 24, pp. 109–165, Jan. 1989. 60, 101

[53] C. J. Date and H. Darwen, *A Guide to the SQL Standard*, vol. 3. Addison-Wesley New York, 1987. 68

[54] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. 77

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014. 77, 78, 79

[56] R. Girshick, "Fast r-cnn," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (USA), p. 1440–1448, IEEE Computer Society, 2015. 77, 78, 80

[57] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, (Cambridge, MA, USA), p. 91–99, MIT Press, 2015. 77, 80

[58] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016. 77

[59] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2017. 77

[60] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, 2017. 77

[61] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016. 77, 81

[62] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. 77, 81

[63] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017. 77, 79

[64] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object detection with deep learning: A review," *arXiv*, vol. 30, no. 11, pp. 3212–3232, 2018. 78, 80

[65] M. Wrenninge and J. Unger, "Synscapes: A photorealistic synthetic dataset for street scene parsing," *CoRR*, vol. abs/1810.08705, 2018. 82

[66] S. R. Richter, Z. Hayder, and V. Koltun, "Playing for benchmarks," *CoRR*, vol. abs/1709.07322, 2017. 82

[67] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *arXiv preprint arXiv:1612.01079*, 2016. 82

[68] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," *CoRR*, vol. abs/1903.11027, 2019. 82

Appendix

Appendix A

Object detection state of the art

The chapter will start analyzing object detection and how synthetic data allows to tackle the problem avoiding the annotation and real data gathering costs.

A.1 Object detection

Object detection is the Computer Vision problem of localizing instances of semantic objects of a certain class such as humans, cars and buildings. We can see two different challenges in this task:

- Labelling each object, which is a classification problem, this is solved as a softmax classification problem, yielding the class scores.
- Localizing each object, usually defined by a bounding box (x, y, w, h)‖(x_0, y_0, x_1, y_1) where x, y represent the box centre and w and h its width and height. Other common representation is x_0, y_0 representing it's upper left corner and x_1, y_1 it's lower right corner.

A.1.1 Main algorithms in deep learning

As mentioned in the introduction of this chapter, object detection handle two tasks, locating and classifying existing objects from one image and labeling them with bounding boxes yielding confidence scores of existing. The frameworks of generic object detection methods can mainly be categorized into two types A.1. One branch follows the traditional object detection pipeline started with R-CNN [54] where region proposals are proposed at first and then classifying each proposal into different object categories. We find a clearly distinguished branch regarding object detection as a regression or classification problem (one shot detectors), choosing a unified framework to achieve final outputs (classification score and bounding boxes) directly. The region proposal-based methods mainly include R-CNN [54],spatial pyramid pooling (SPP)-net [55], Fast R-CNN [56],Faster R-CNN [57], region-based fully convolutional network(R-FCN) [58], feature pyramid networks (FPN) [59], and Mask R-CNN [60], some of which are correlated with each other (e.g., SPP-net modifies R-CNN with an spatial pyramid pooling layer).The one-shot algorithms mainly include YOLO [61],Single Shot MultiBox Detector (SSD) [62], and Retina-net [63]. The idea of anchors introduced in Faster R-CNN bridges this two branches. Details of these methods are as follows.

Region based CNN detectors:

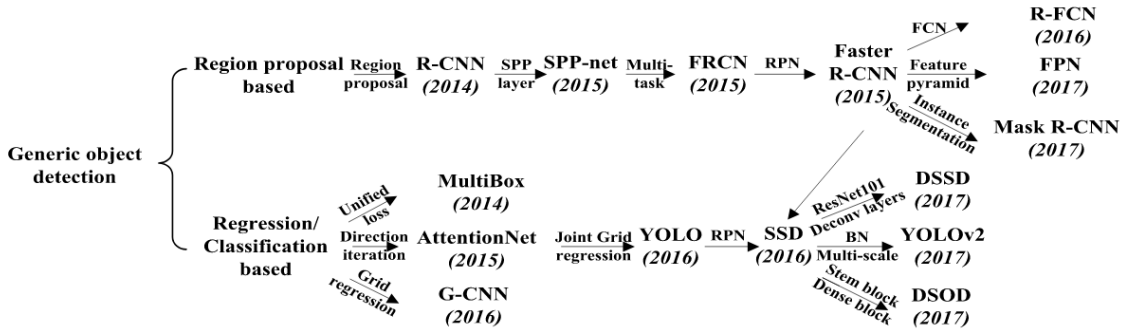


Figure A.1: Frameworks of generic object detection methods from [64].

1. R-CNN: The R-CNN adopts selective search to generate about 2000 region proposals for each image, then each proposal it's wrapped to fit a CNN which computes a 4096-dimensional feature vector. With pre-trained category-specific linear SVMs for multiple classes, each feature vector is scored. The scored regions are then adjusted with bounding box regression and filtered with a greedy nonmaximum suppression (NMS) to produce final BBs for preserved object locations.

2. SPP-net: Follows the work of the previous net solving the main problem of the R-CNN regions proposals may include partly the object, and the warping operation produces unwanted geometric distortions. The solution is comes from spatial pyramid pooling layer, which takes the features from the fifth layer of the net and creates a pyramid of the feature vector A.2

3. Fast R-CNN: Taking the SPP layer this architecture processes the whole image to produce feature maps. Then, a fixed length feature vector is extracted from each region proposal and processed with a RoI pooling layer (a SPP layer with just one pyramid level). Deviating from the previous works the bounding boxes and confidences are computed from fully connected layers A.4.1

4. Faster R-CNN: Improves it's predecessor removing region proposals computation which was a bottleneck in improving efficiency. Introducing a new region proposal network, RPN A.5.1, takes an image of arbitrary size to generate a set of rectangular object proposals. RPN takes as input the features from a specific conv layer with the preceding layers being shared with the object detection network.

5. FPN: Feature pyramids had been widely applied in many object detection systems to improve scale invariance [55] A.6.1(a) however training time and memory consumption were not feasible. Previous approaches took only a single input scale to represent high-level semantics and increase the robustness to scale changes A.6.1(b), other approaches had image pyramids built at test time leading results to inconsistencies between train/test-time inferences [56]. Improving said techniques FPN takes a top-down (TD) pathway and several lateral connections to combine low-resolution and higher hierarchical features with high-resolution and lower hierarchical features A.6.1(d). The bottom up path is obtained by a $3x3$ convolution with a stride of 2, the top down path is the result of upsampling the upper lever features and convolving them with a $1x1$ filter, then an element-wise

addition is performed with the same level features from the bottom-up pyramid.

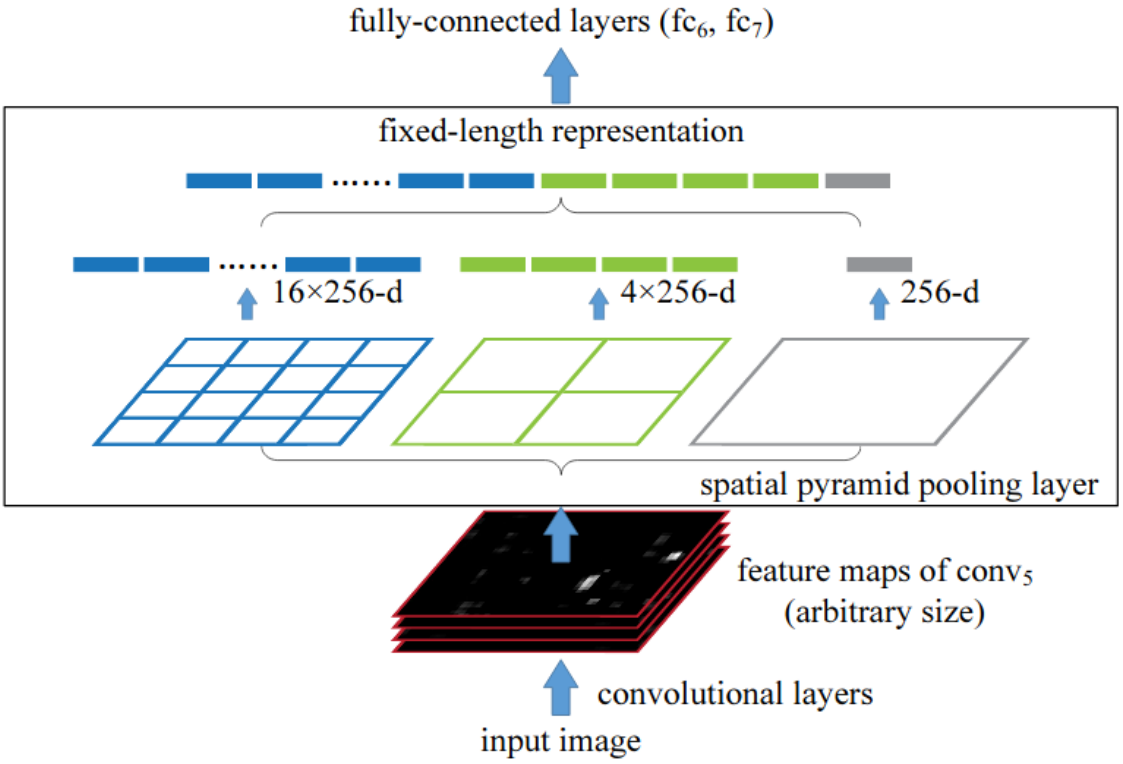


Figure A.2: Spatial pyramid pooling from [55].

One-stage detectors

1. YOLO: This work proposes to divide the input image into an S × S grid. Each grid cell predicts the whether or not there is an object within it's boundaries. It outputs bounding boxes and their corresponding confidence scores and conditional class probabilities A.8.1.

2. SSD: Following the line of work from YOLO, improving its capacity to deal with small objects in groups and new aspect ratios, by adding several feature layers to the end of the network A.9.1.

3. Retina Net: Follows the FPN TD architecture 5, it's novelty comes from the focal loss introduction [63], which lowers the loss weight for well classified samples and increases the loss for difficult ones. This is important due to the object detection being a heavily unbalanced problem, most grid cells wont contain objects and one-stage detectors in contrast with region proposal detectors analyze all possible region candidates.

A.1.2 Synthetic data in object detection

After describing the main algorithms in object detection we will follow our analysis with how synthetic data has been applied to yield better results, we will use as an

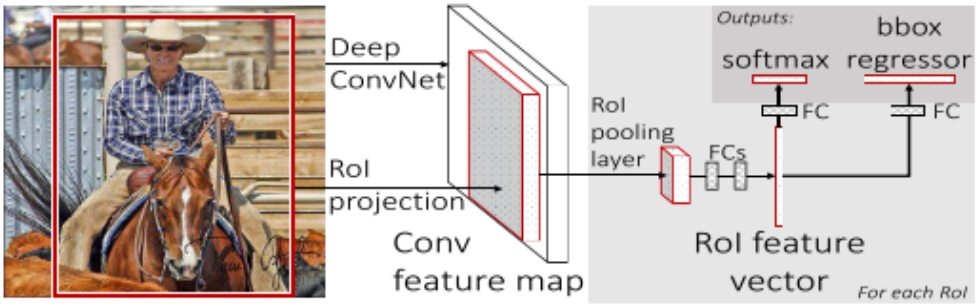


Figure A.3: Fast R-CNN from [56].

(A.4.1)

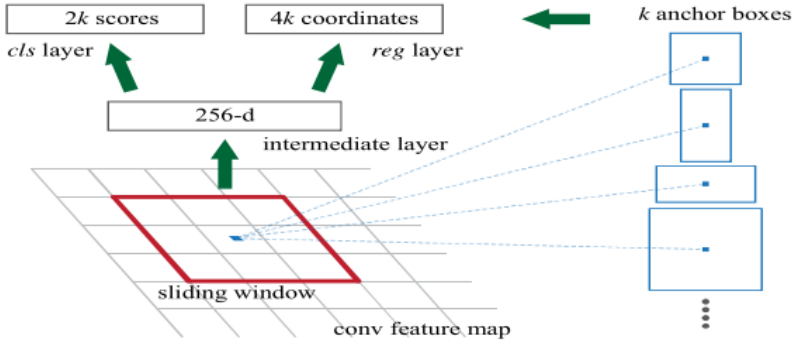


Figure A.4: RPN from [57].

(A.5.1)

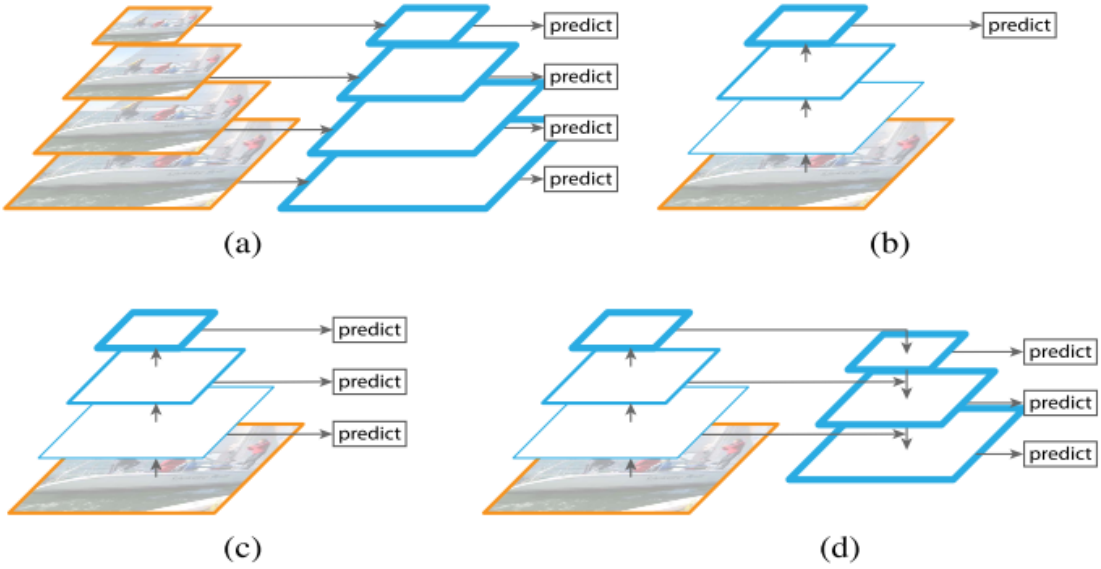


Figure A.5: (a) Slow to use an image pyramid to build a feature pyramid. (b) Only single-scale features are adopted for faster detection. (c) Reuse the pyramidal feature computed by a ConvNet. (d) FPN integrates both (b) and (c). Blue outlines indicate feature maps and thicker outlines denote semantically stronger features. Pyramid features from [64].

(A.6.1)

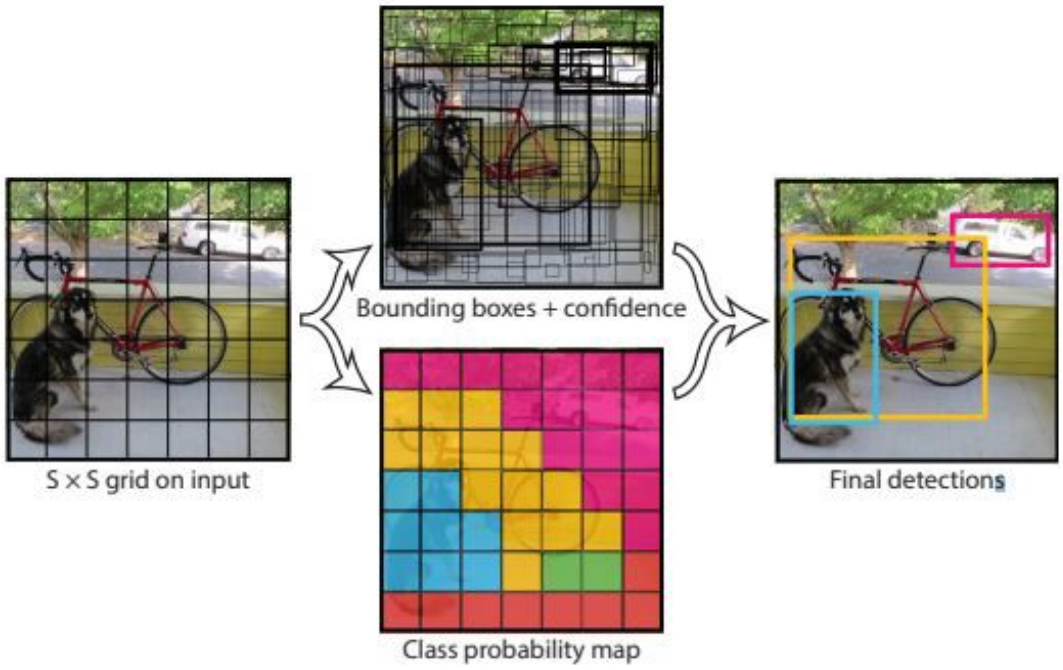


Figure A.7: Yolo from[61]

(A.8.1)

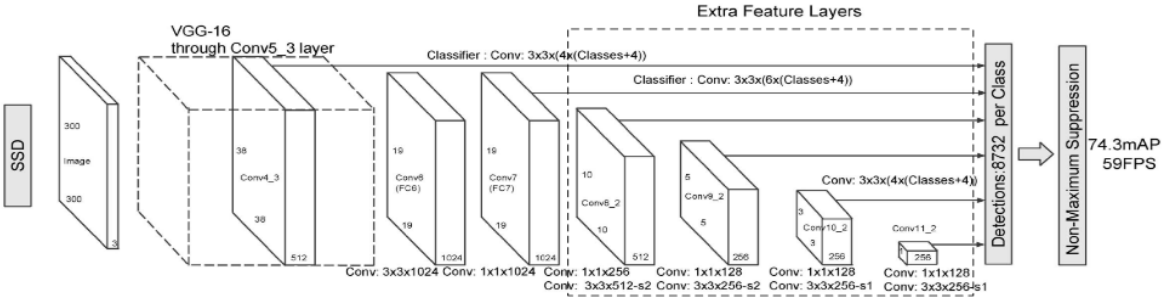


Figure A.8: SSD from[62]

(A.9.1)

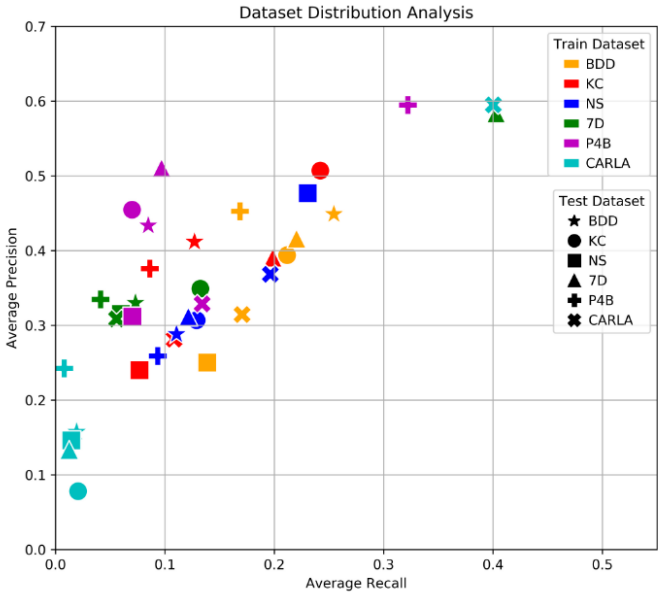


Figure A.10: Results of using synthetic data from scratch from[17]

starting point the results from [17] which studies how different algorithms are able to generalize from only synthetic data, fine tuning with real data and different mixtures of real and synthetic data. Then we will see how can we improve this results by binding real and synthetic data from [18] and [9] and we will conclude with an study into curriculum learning with synthetic data and a comparison between results with [19].

When training with synthetic data the firsts questions which rise are how can we avoid the assumption that the source and target domains share the same characteristics in a transformed feature space?, how can we handle the basic assumption in conventional machine learning that the training and test data are sampled independently from an identical distribution, or *i.i.d* assumption in short?
Figure A.10 showcases the results of training with a synthetic distribution such as 7D (Synscapes dataset [65]), P4B (Playing for Benchmark [66]) or CARLA [41] and testing on a real data such as BDD (Berkeley Deep Drive dataset [67]), KC (Kitti-CityScapes [36]) or NS (NuScenes [68]). We can extrapolate that synthetic datasets suffer from a specificity problem which results in models that are incapable of proper generalization. They perform very well on their own test set, however their performance suffers on any other test sets. However something we can infer from this paper graph is that while the domain gap between synthetic and real data seems broader than the one found between real datasets, there's also a domain discrepancy between real datasets, yielding almost half the performance on another real test set than on their original.

The paper follows trying different approaches such as training with a mixture of synthetic and real data and fine tuning with real data A.11, we can see how fine tuning with real data yields much better results than mixing a subset of real data with the synthetic data. Finally the paper illustrates how with fine tuning with only a 10% of the real data and using much more synthetic data (using all synthetic data sets for training) we can achieve better results than with the full original train set A.12

Following the great results mentioned, we will analyze paths to improve the training with synthetic data, two approaches will be mentioned:

- Domain randomization [9]: authors exploit the opportunities of synthetic data

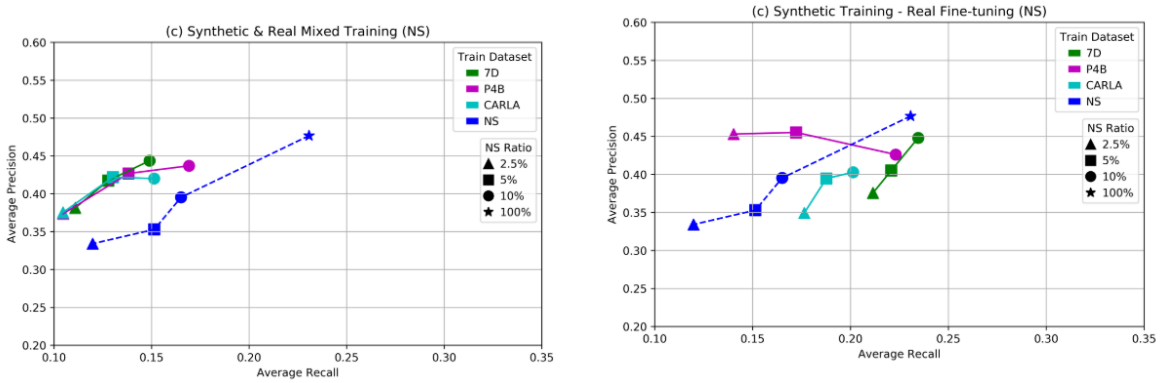


Figure A.11: Results of using a mixture of real and synthetic data for training and using only synthetic data and then fine tuning. [17]

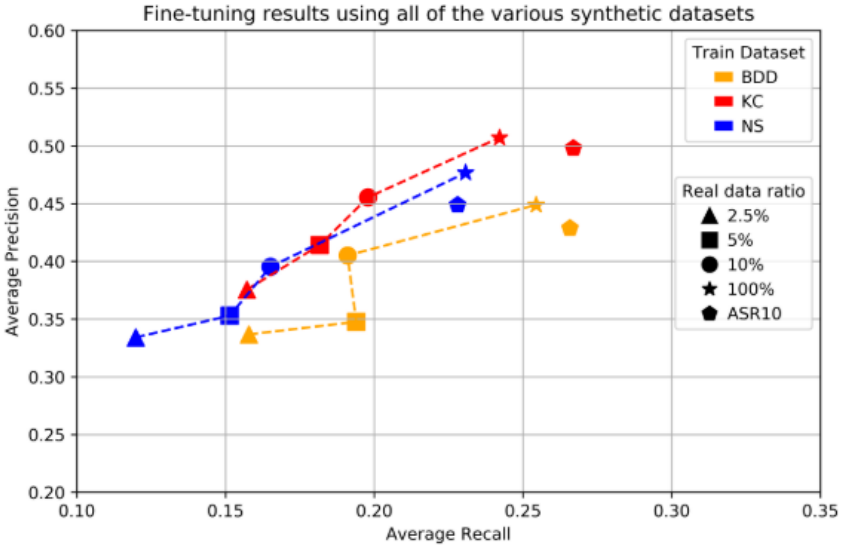


Figure A.12: Results of using all synthetic datasets and fine tuning with a 10% of the real dataset (3% of train size). The results are shown as ASR10. [17]

to generate instances of the objects in random positions, on top of a random background along with random flying distractors (geometric shapes next to the background images) in a scene with random lighting from random viewpoints. Before rendering, random texture is applied to the objects of interest as well as to the flying distractors. The resulting images, along with automatically-generated ground truth (right), are used for training a deep neural network, see A.13. The obtained results improve the ones of synthetic datasets A.14, although the domain randomization dataset is 40 times larger than the synthetic dataset.

- Curriculum learning for object detection [19]:This second approach follows the idea of curriculum learning with synthetic samples. The authors propose to measure the complexity of a sample by the scale and position of the 3d model in the image. By overlaying synthetic model in random positions of original scenes the net can learn to detect the object. The pacing for the curriculum learning is determined by the scale, the bigger the object the easier it should be, therefore they propose to teach the net increasingly harder samples in order to exploit the full potential, see Figure A.16.1. The authors conclude their analysis by comparing the performance of the model when trained only by this approach and by using different amounts of real images, see Figure A.17.1.

Both approaches although different showcase how training with synthetic data yields models that compete favorably with object detectors trained purely on real images.

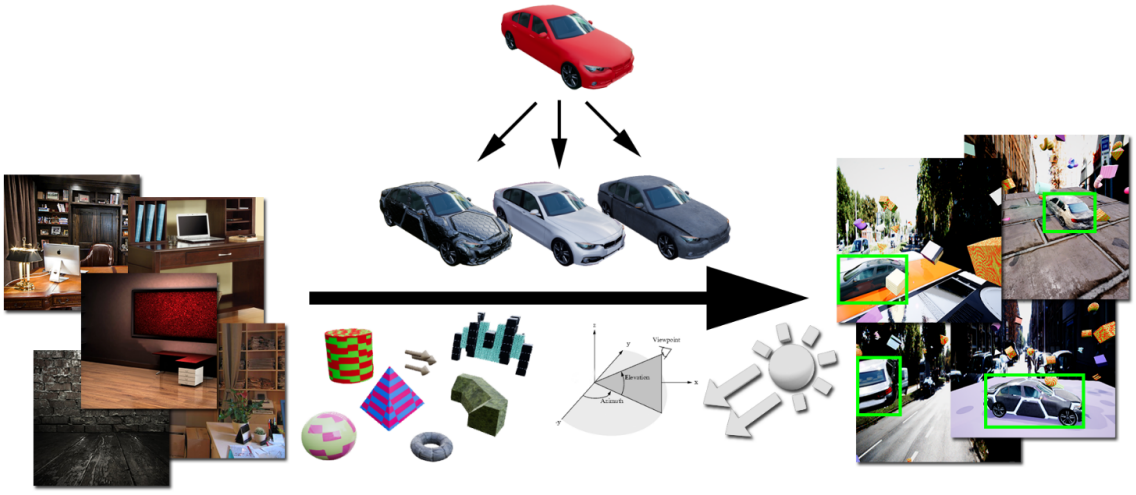


Figure A.13: Process of domain randomization. [9]

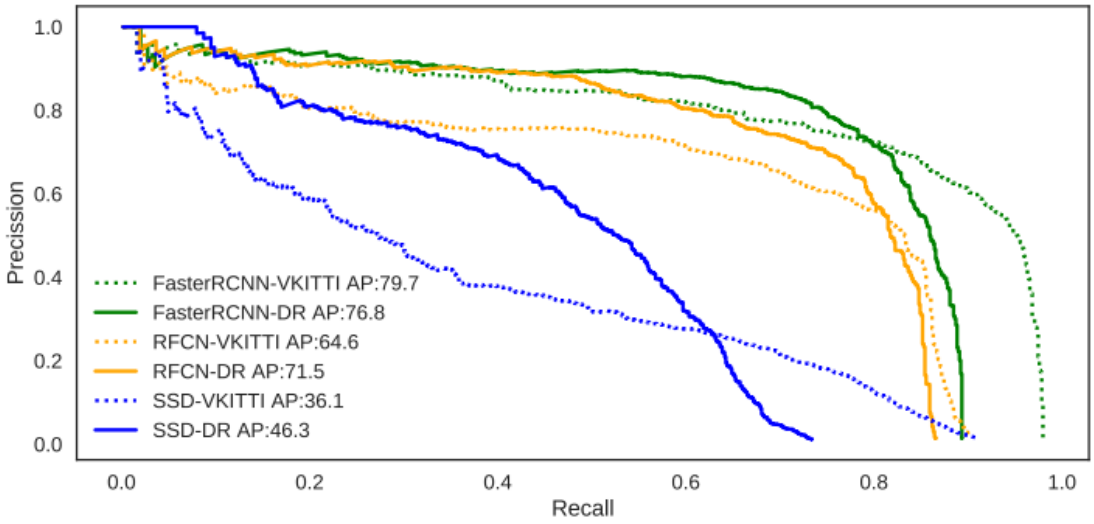


Figure A.14: Results of domain randomization vs VKITTI. [9]

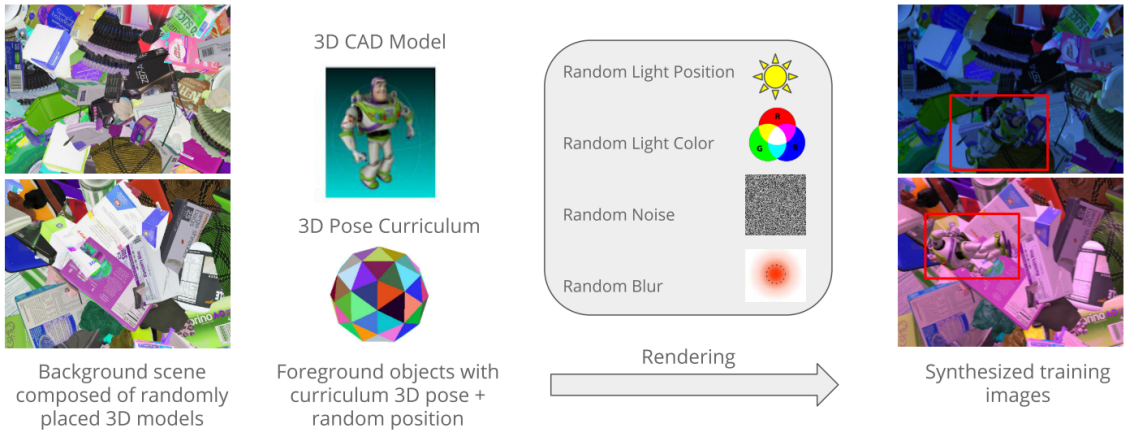


Figure A.15: Curriculum learning in object detection with synthetic images. [19]

(A.16.1)

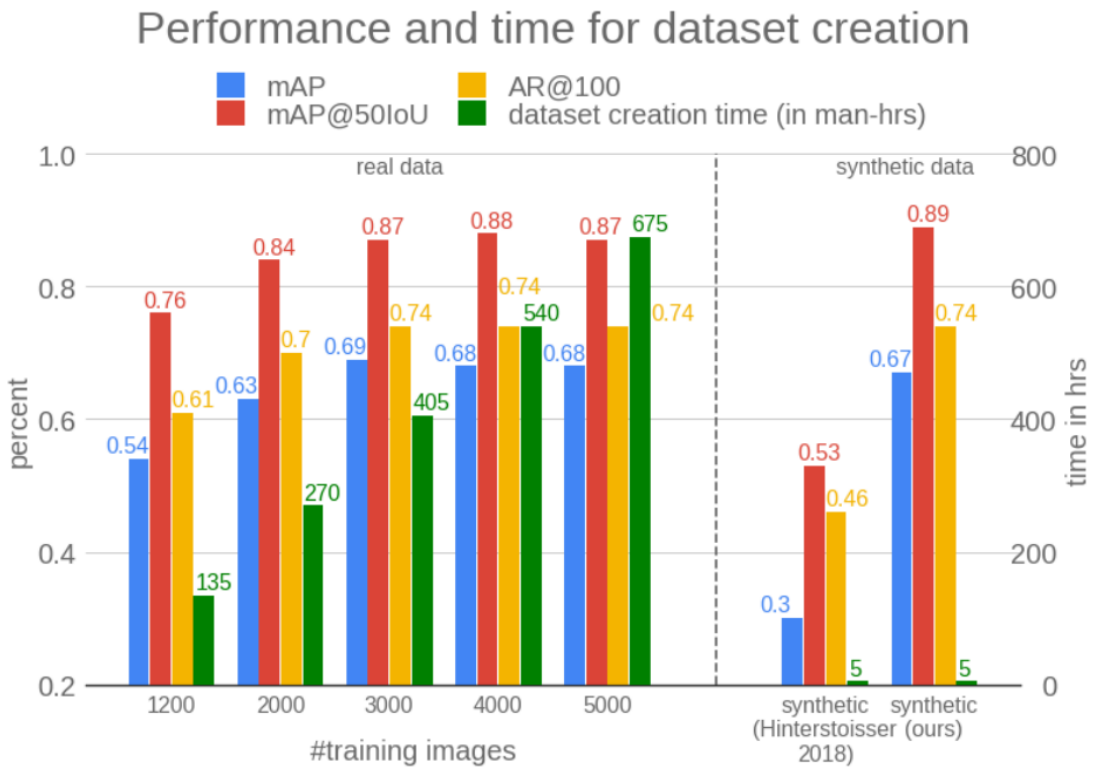


Figure A.16: The authors compared not only the performance in MAP(blue bar), MAP when considering a match $IoU \geq 0.5$(red bar) and average recall at 100 detection candidates (yellow bar), but also the amount of man hours (green bar) to create the dataset. [19]

(A.17.1)

A.1.3 Conclusions from synthetic data applied to object detection

In this section we summarize the insights gathered from the previously mentioned state of the art into three main ideas:

- Domain shift occurs between real data: From Figure A.10 we can extrapolate that although the best results when there's no mixture between domains are achieved when training with the original data, when training with synthetic data such as 7D (synthetic) and testing on KC(real), green circle, yields better results than training with NS(real), blue circle. This means that the domain gap persist even among real datasets and domain adaptation is needed nevertheless.

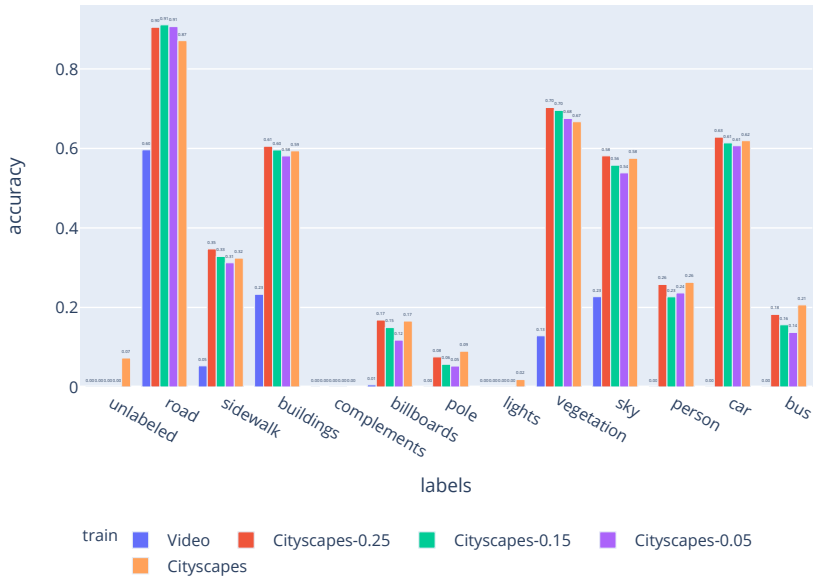
- Training with real data a previously trained network with synthetic data outperforms training with just real data: From Figure A.12 we can see how consistently training using pretrained weights on synthetic data and finetuning with a the real dataset yields better results than training with only real data. Furthermore, we see how training with as little as 10% of the real data can achieve similar results than with the whole dataset.

- When having access to unlimited amounts of synthetic data we can train smarter, yielding models which can compete with the ones trained with real data: Figure A.17.1 shows the results obtained when using curriculum learning on purely synthetic data yields better results than a synthetic data overlayed on real data, and comparable results to training with up to 5000 real images.

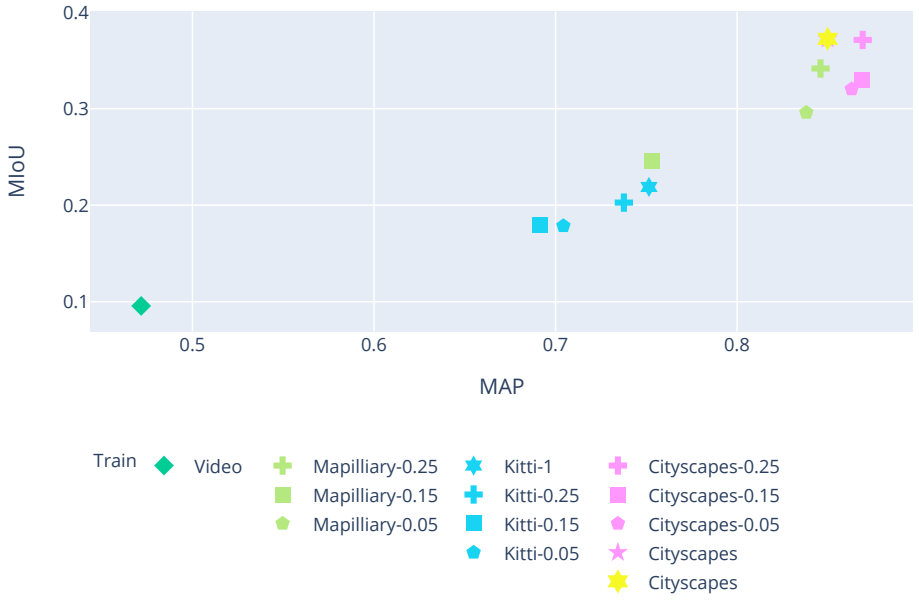
Appendix B

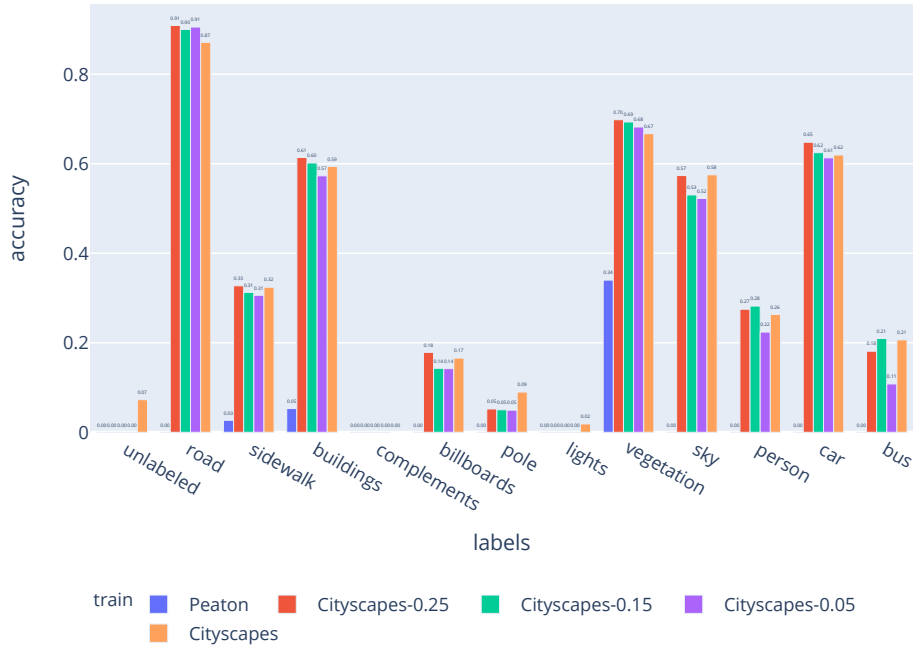
Hybrid train sets from synthetic and real sequences Performance.

B.1 Mixed with Cityscapes, tested on Cityscapes

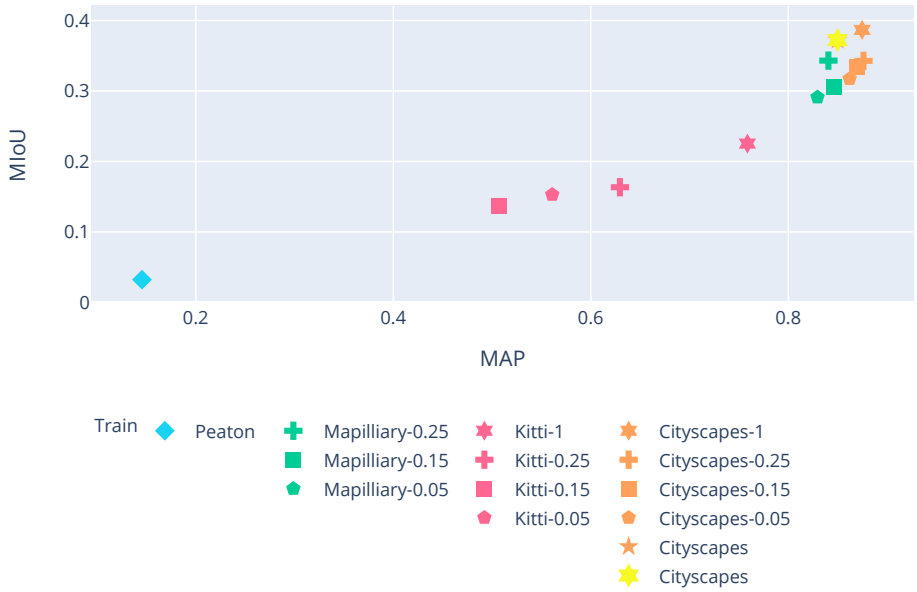


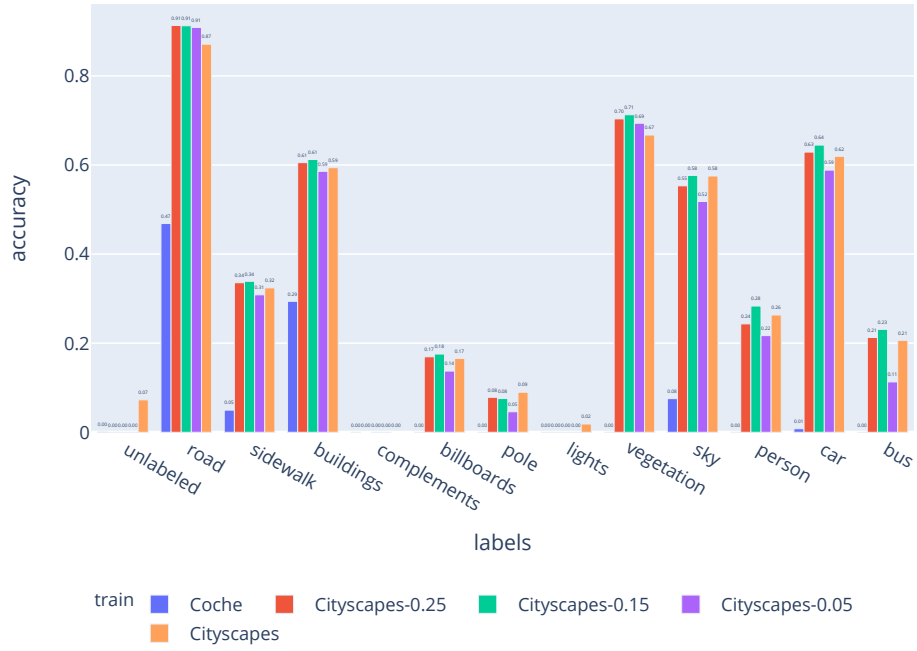
Static


MIoU
0.4
0.3
0.2
0.1
0.5
0.6
0.7
0.8
MAP
Train
Video
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Cityscapes
Cityscapes

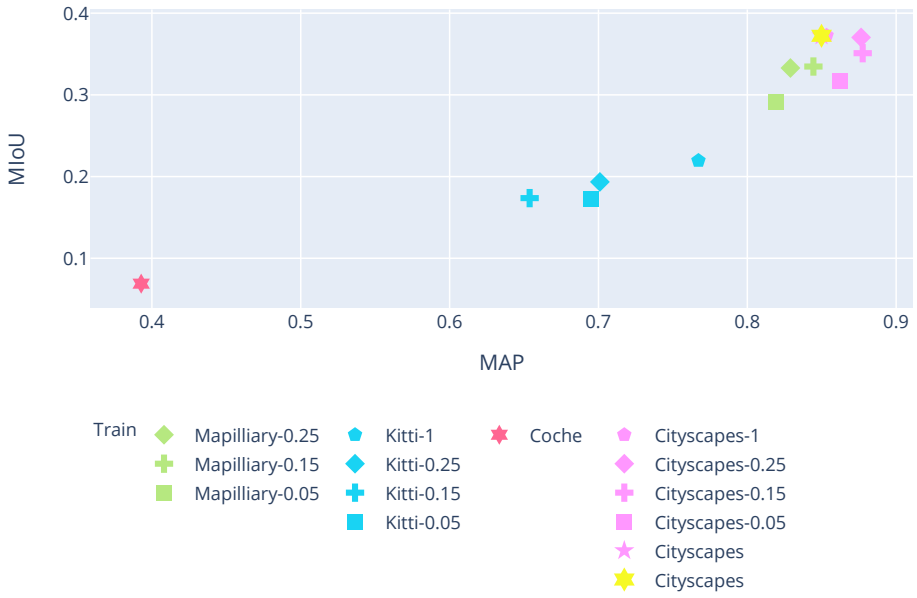

accuracy
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Peaton
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Cityscapes

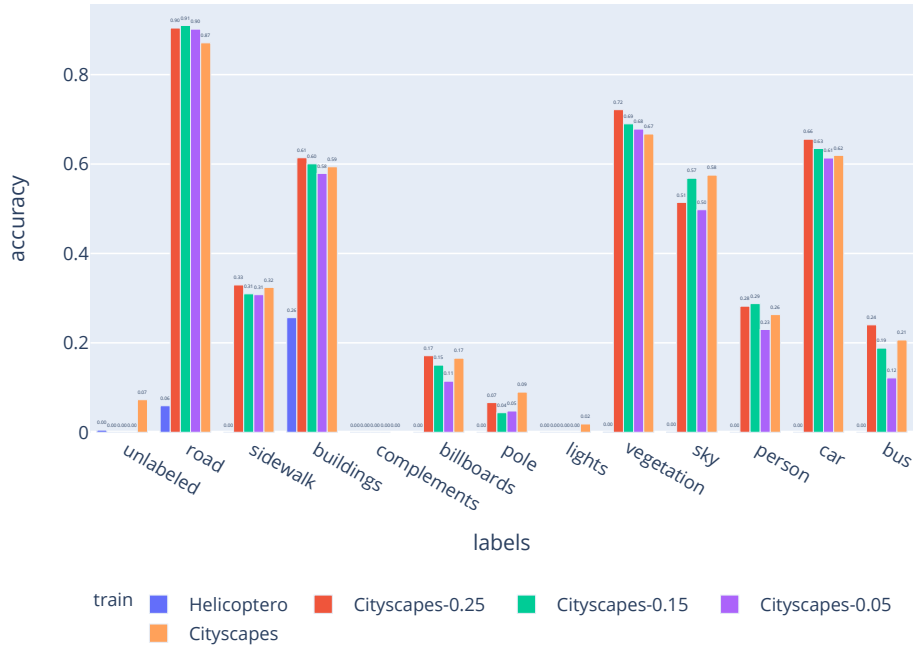
Pedestrian


MIoU
MAP
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8
Train
Peaton
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Cityscapes-1
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Cityscapes
Cityscapes

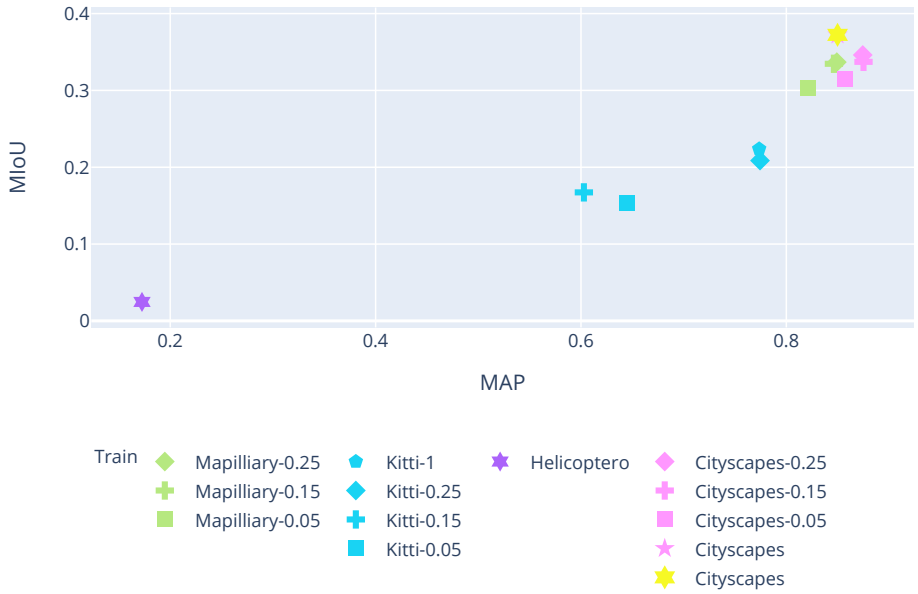

accuracy
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Coche
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Cityscapes

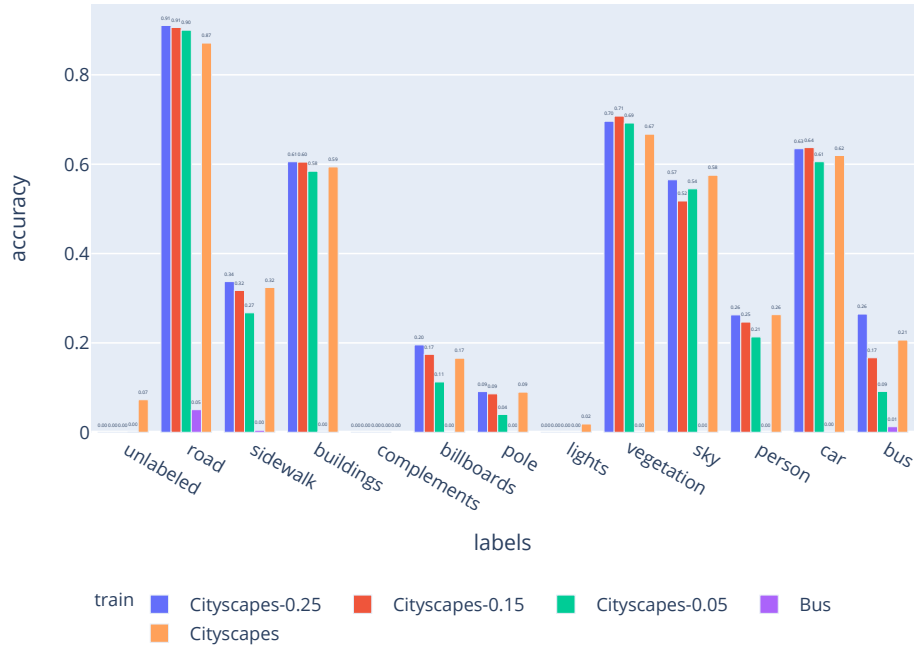
Car


MIoU
0.4
0.3
0.2
0.1
0.4
0.5
0.6
0.7
0.8
0.9
MAP
Train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Coche
Cityscapes-1
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Cityscapes
Cityscapes


accuracy
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Helicoptero
Cityscapes
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05

Helicopter


MIoU
MAP
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8
Train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Helicoptero
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Cityscapes
Cityscapes


accuracy
labels
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
train
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05
Bus
Cityscapes

Bus

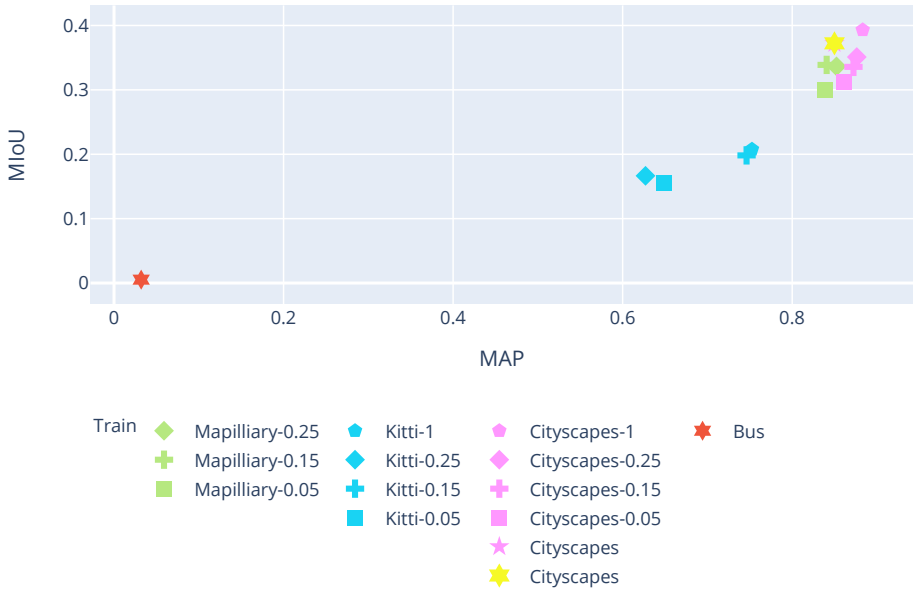
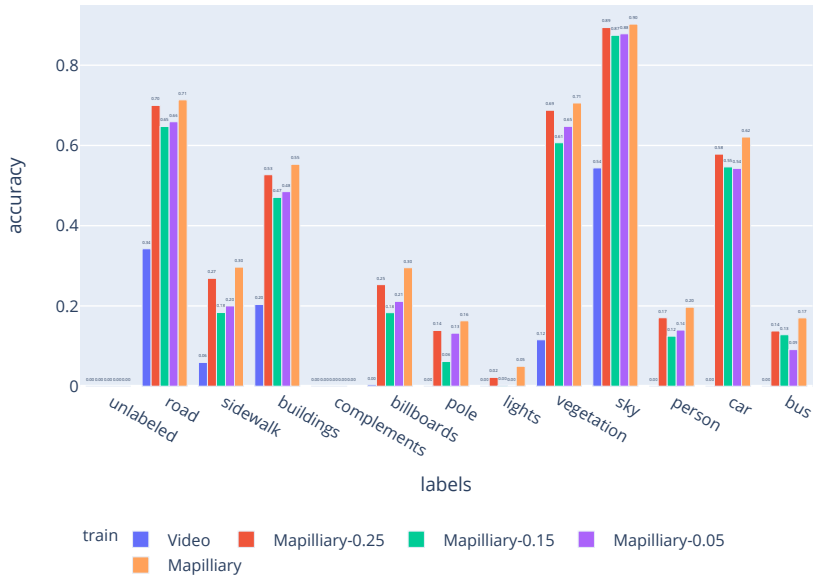
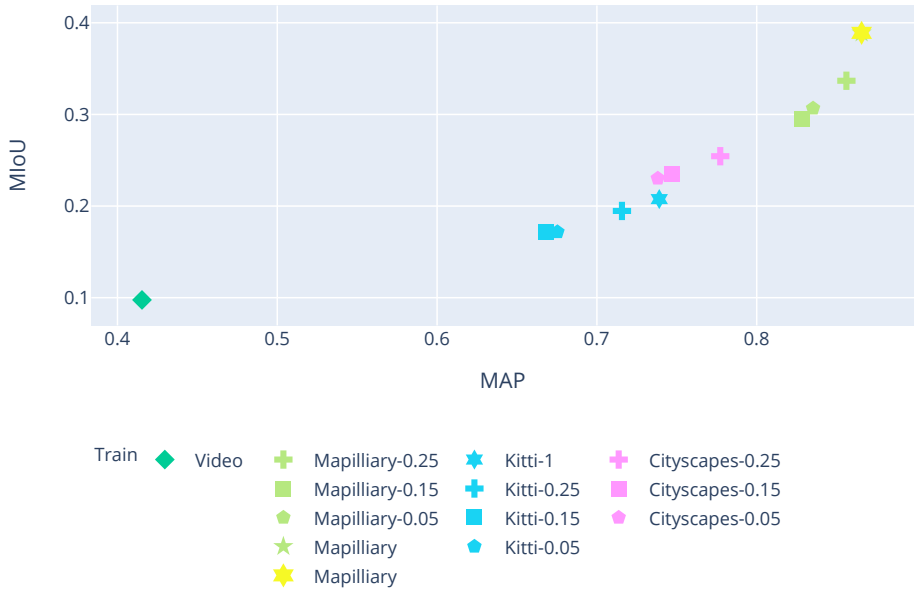


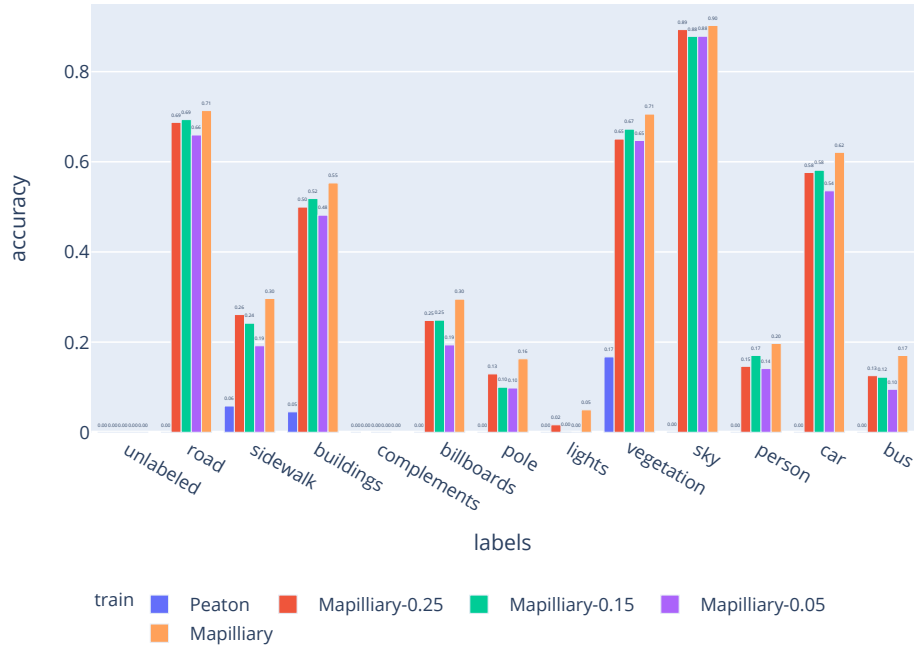
Table B.1: Results obtained from a deeplabv3 trained from scratch with a mixture of each of the synthetic video sequences and a fixed fraction of the Cityscapes dataset.

B.2 Mixing with Mapilliary, testing on Mapilliary

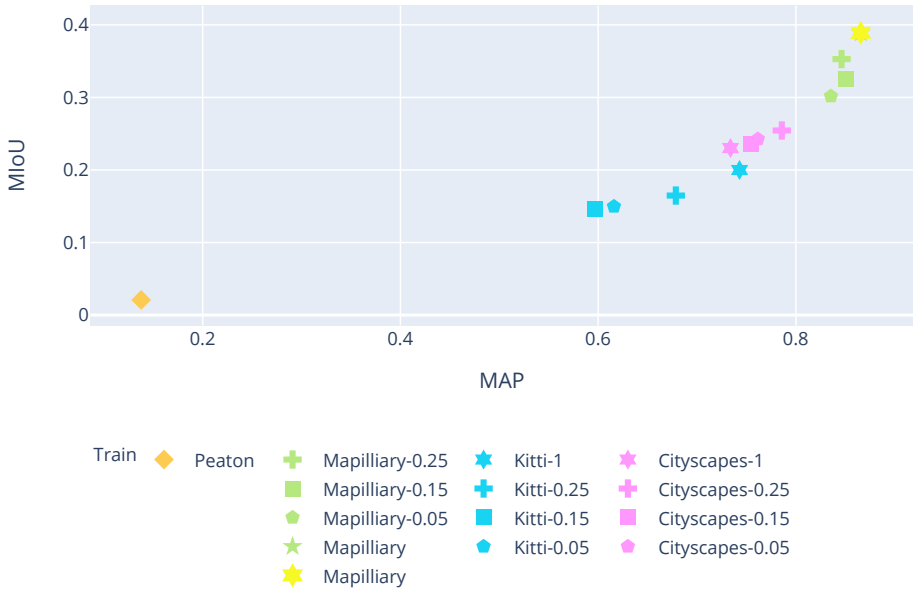


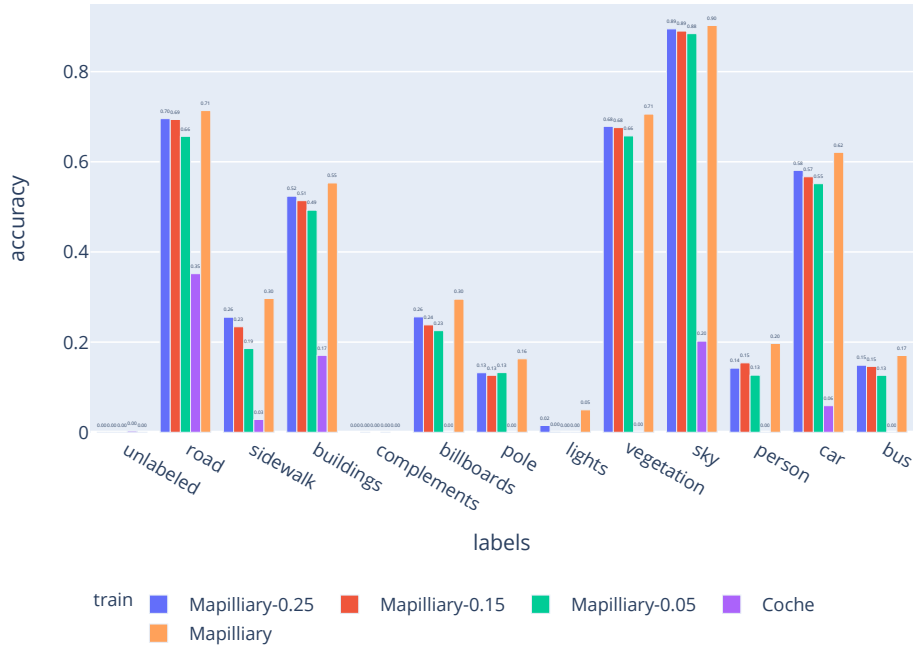
Static


MIoU
0.4
0.3
0.2
0.1
0.4
0.5
0.6
0.7
0.8
MAP
Train
Video
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Mapilliary
Mapilliary
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05

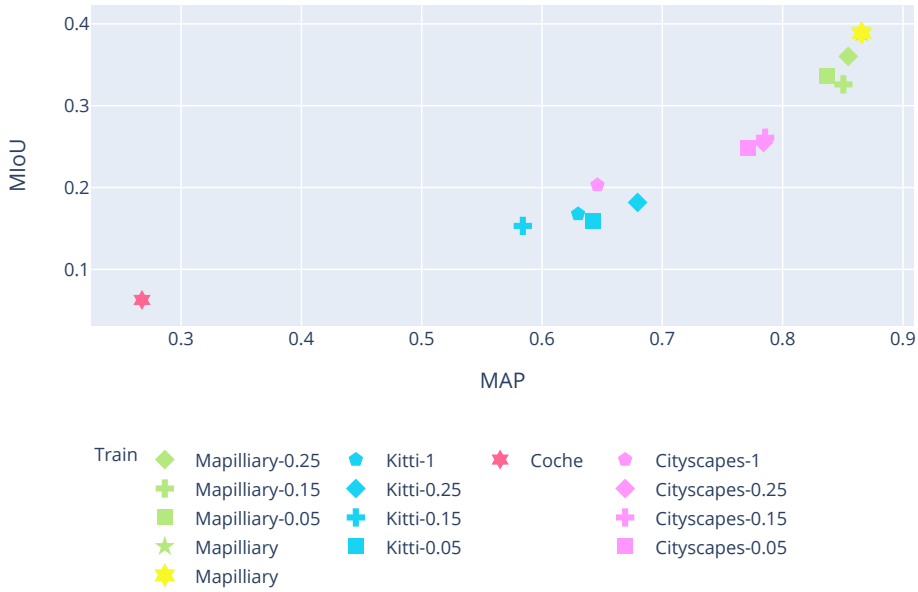

accuracy
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Peaton
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Mapilliary

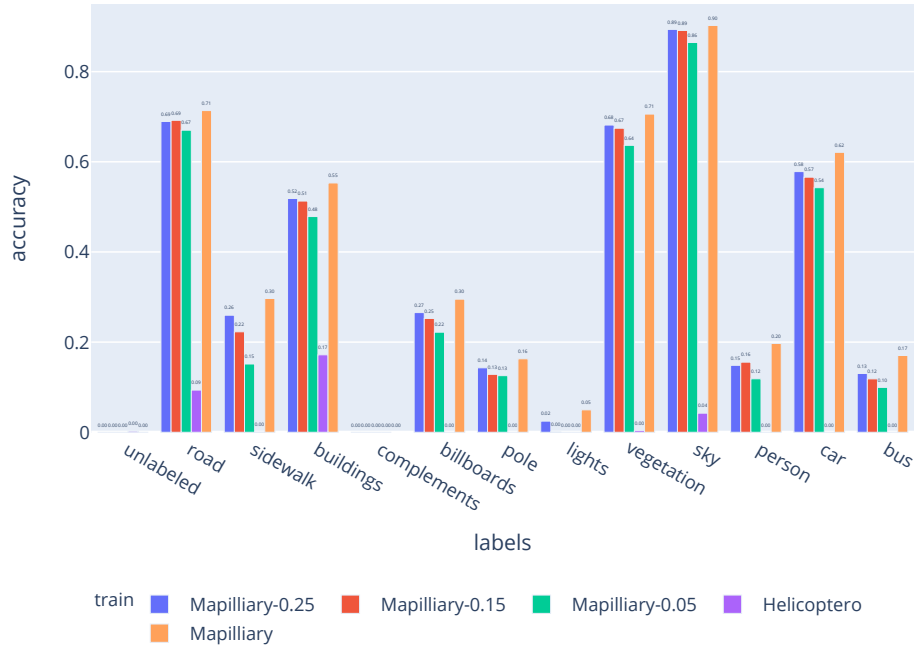
Pedestrian


MIoU
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8
MAP
Train
Peaton
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Mapilliary
Mapilliary
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Cityscapes-1
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05

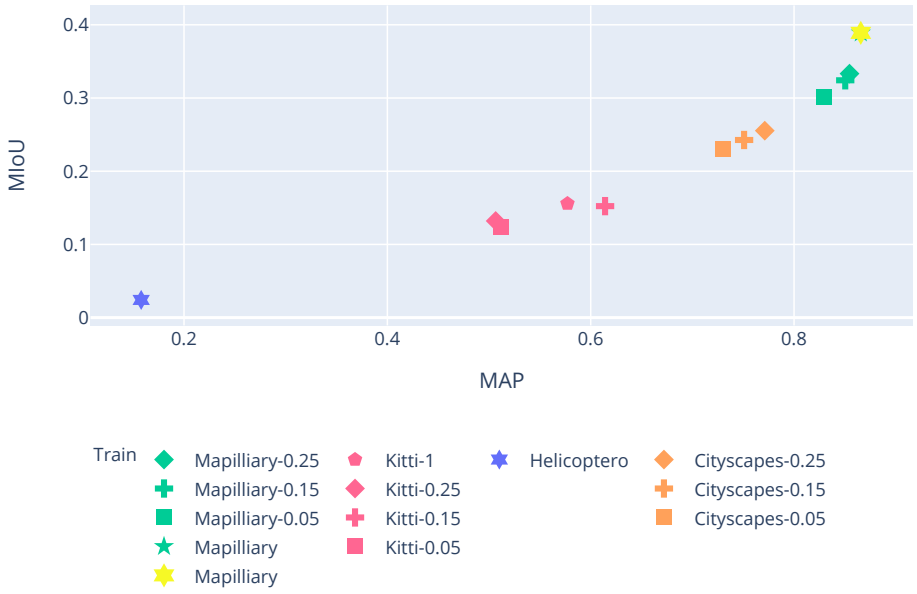

accuracy
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Coche
Mapilliary

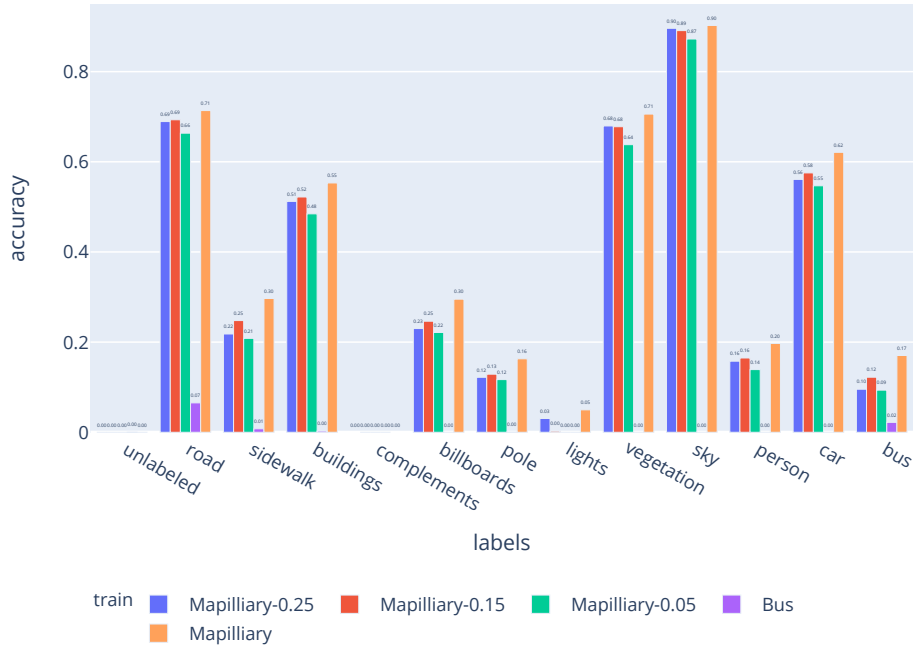
Car


MIoU
MAP
0.1
0.2
0.3
0.4
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Mapilliary
Mapilliary
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Coche
Cityscapes-1
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05


accuracy
0
0.2
0.4
0.6
0.8
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Helicoptero
Mapilliary

Helicopter


MIoU
0.4
0.3
0.2
0.1
0
0.2
0.4
0.6
0.8
MAP
Train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Mapilliary
Mapilliary
Kitti-1
Kitti-0.25
Kitti-0.15
Kitti-0.05
Helicoptero
Cityscapes-0.25
Cityscapes-0.15
Cityscapes-0.05


accuracy
0.8
0.6
0.4
0.2
0
unlabeled
road
sidewalk
buildings
complements
billboards
pole
lights
vegetation
sky
person
car
bus
labels
train
Mapilliary-0.25
Mapilliary-0.15
Mapilliary-0.05
Bus
Mapilliary

Bus

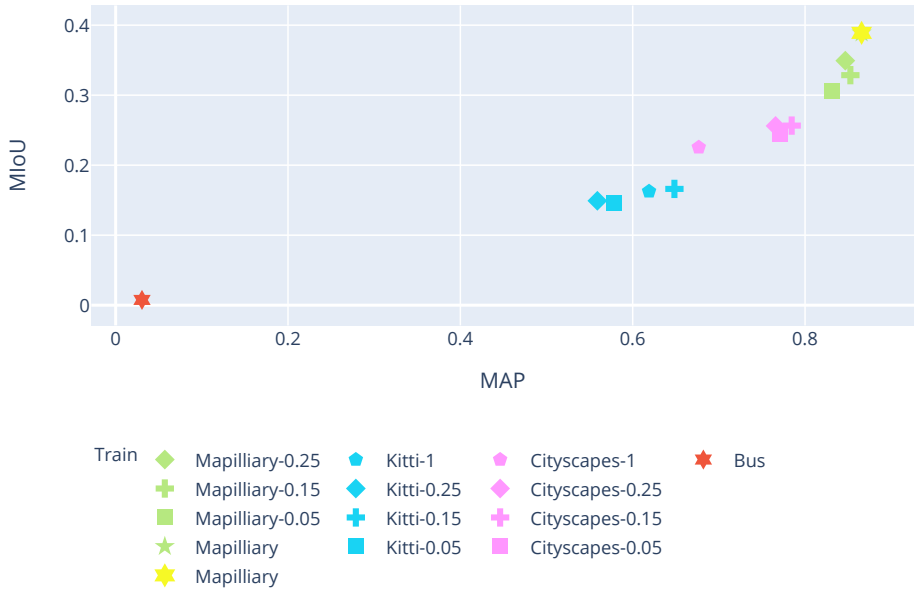


Table B.2: Results obtained from a deeplabv3 trained from scratch with a mixture of each of the synthetic video sequences and a fixed fraction of the Mapilliary dataset.

Appendix C

Catastrophic forgetting when pretrained on real datasets.

In the following Figures C.1, C.2,C.3, we will include the progress of the MAP and MIoU when finetuning pretrained models with a portion of Cityscapes and the Mapilliary datsets. The results are obtained by testing on the Cityscapes, Mapilliary and Kitty test sets.

Furthermore, we can see how when finetuning the pretrained model on the Kitty dataset, the model forgets the knowledge of obtained through it's training, therefore loosing accuracy on the Kitty dataset and gaining accuracy on the other test sets. Although it improves performance after transfer learning to other sets, we can see how consistently the models pretrained on synthetic datasets yield better general results than the one from the Kitti set. This problem was first introduced in [52], where they explain how the information is distributed through the weights of the network and learning a new task or the same but in a new domain alters the weights and connections which were involved into the inference of the original task explaining the drop in performance on the original task.

We believe that due to the amount of images and variability synthetic datasets provide, bring an unrivaled power of abstraction when transfer learning techniques are used.

Figure C.1: Finetuning with 5% of the Cityscapes dataset

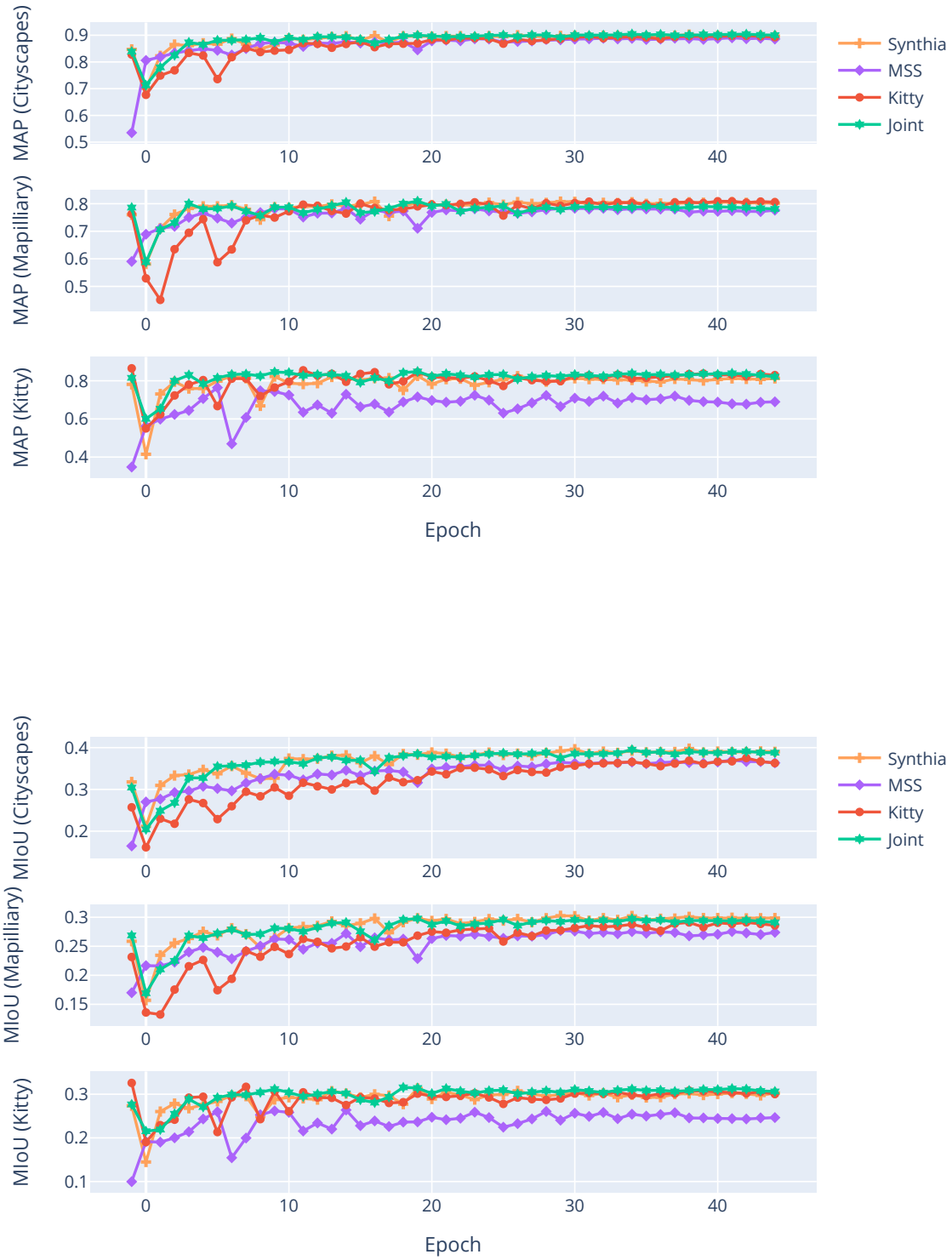
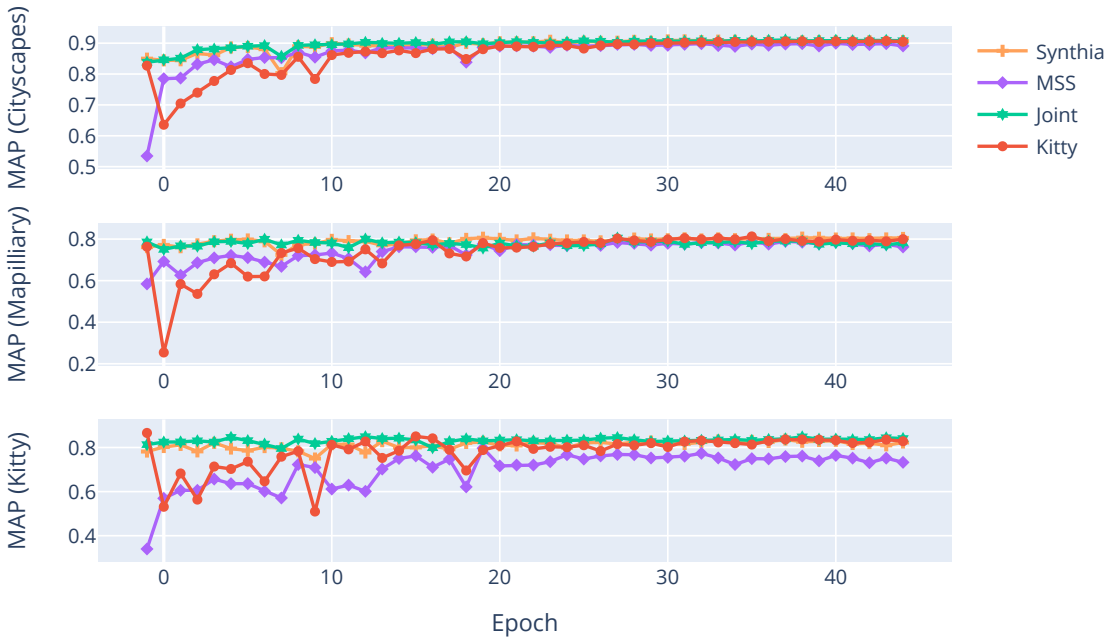


Figure C.2: Finetuning with 15% of the Cityscapes dataset



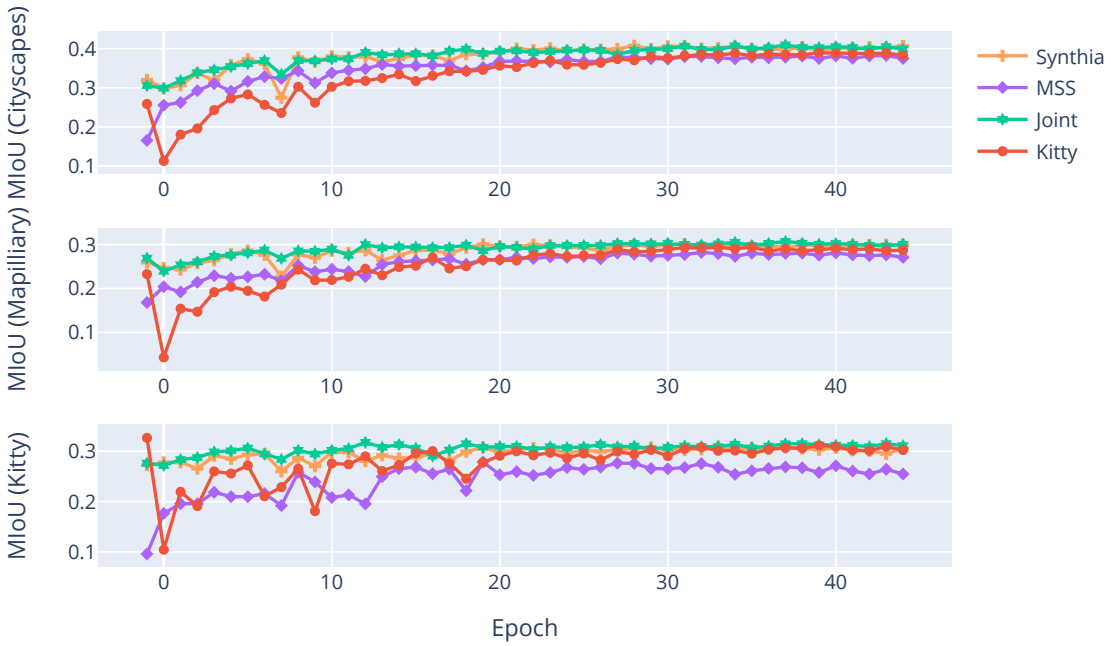


Figure C.3: Finetuning with 25% of the Cityscapes dataset

Appendix D

Sequence diagram

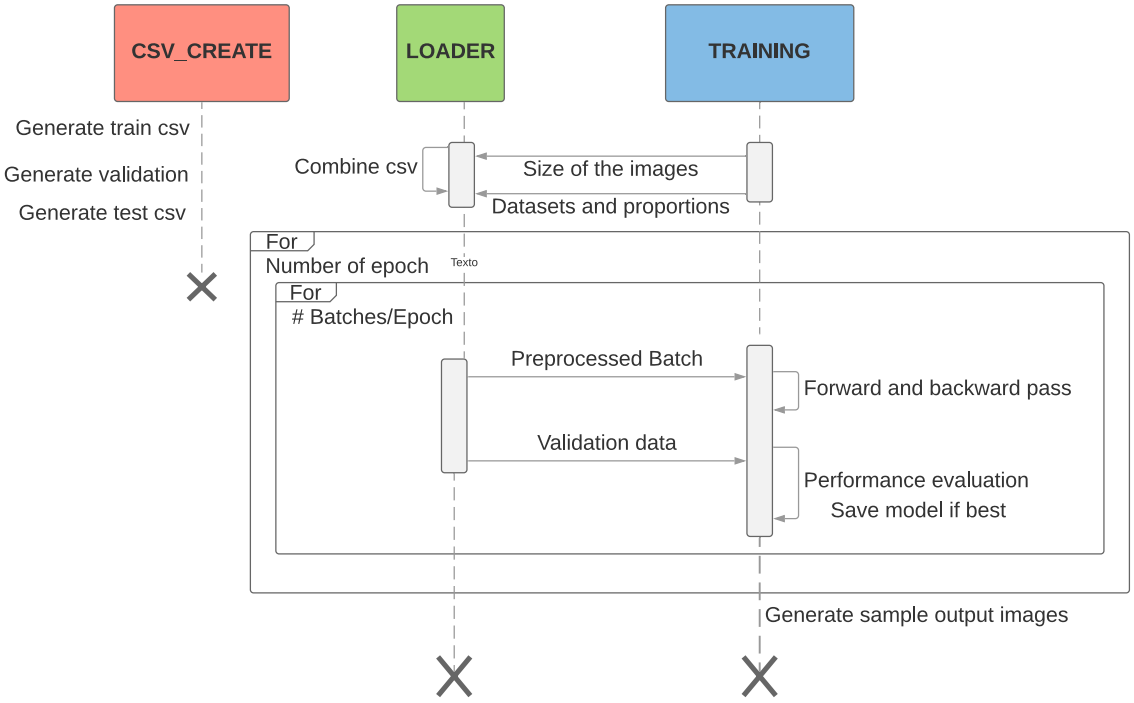


Figure D.1: Sequence diagram